# SCA-LDPC: A Code-Based Framework for Key-Recovery Side-Channel Attacks on Post-Quantum Encryption Schemes

Qian Guo[1], Denis Nabokov[1], Alexander Nilsson[1,2], and Thomas Johansson[1]

[1] Dept. of Electrical and Information Technology, Lund University, Lund, Sweden
{qian.guo,denis.nabokov,alexander.nilsson,thomas.johansson}@eit.lth.se
[2] Advenica AB, Malmö, Sweden

**Abstract.** Whereas theoretical attacks on standardized crypto primitives rarely lead to actual practical attacks, the situation is different for side-channel attacks. Improvements in the performance of side-channel attacks are of utmost importance.
In this paper, we propose a framework to be used in key-recovery side-channel attacks on CCA-secure post-quantum encryption schemes. The basic idea is to construct chosen ciphertext queries to a plaintext checking oracle that collects information on a set of secret variables in a single query. Then a large number of such queries is considered, each related to a different set of secret variables, and they are modeled as a low-density parity-check code (LDPC code). Secret variables are finally determined through efficient iterative decoding methods, such as belief propagation, using soft information. The utilization of LDPC codes offers efficient decoding, source compression, and error correction benefits. It has been demonstrated that this approach provides significant improvements compared to previous work by reducing the required number of queries, such as the number of traces in a power attack.
The framework is demonstrated and implemented in two different cases. On one hand, we attack implementations of HQC in a timing attack, lowering the number of required traces considerably compared to attacks in previous work. On the other hand, we describe and implement a full attack on a masked implementation of Kyber using power analysis. Using the ChipWhisperer evaluation platform, our real-world attacks recover the long-term secret key of a first-order masked implementation of Kyber-768 with an average of only 12 power traces.

**Keywords:** Lattice-based cryptography, code-based cryptography, side-channel attacks, NIST post-quantum cryptography standardization, low-density parity-check codes.

## 1 Introduction

NIST [1] is running a standardization process (referred to as the NIST PQ project) for post-quantum public-key cryptographic algorithms (PQC schemes), which are supposed to be secure even against attacks from quantum computers.

This is not the case for most public-key algorithms in use today [42]. The project started in 2017 and just recently the first choices for standardization were announced. The project is ongoing and round 4 will involve a further examination of additional schemes. All of the schemes in the NIST PQ project are based on a variety of hard problems that are believed to be intractable for quantum computers, and many of them can be categorized as either Public-key Encryption (PKE) or Key Encapsulation Mechanisms (KEMs). These PKE/KEM schemes are based on either the Learning with Errors (LWE) problem as introduced by Regev [36] in 2005 or on code-based problems, initiated in [27].

Two such schemes will be considered in this paper. One is CRYSTALS-Kyber [40], selected by NIST as the candidate for standardization for KEMs. The security of Kyber is based on the Module LWE problem and has strong confidence in its theoretical security, while also offering a good performance. The other scheme is HQC [2], a code-based round 4 candidate. Other code-based round 4 candidates are BIKE [4] and Classic McEliece [3]. NIST has stated that one of the schemes HQC or BIKE may be standardized.

LWE- or code-based PKE/KEMs are usually built to be secure against chosen plaintext attacks (IND-CPA secure) and then transformed to be secure against adaptive chosen ciphertext attacks (IND-CCA secure) by applying some CCA conversion method, such as the Fujisaki-Okamoto (FO) transform. The FO transform involves a re-encryption after decryption, which enables the detection of invalid ciphertexts and correspondingly return failure. Invalid chosen ciphertexts that are not proper encryptions of a message will almost always be rejected by the decryption/decapsulation.

Side-Channel Attacks (SCA) were introduced by Kocher [26] and are a separate area of research today. For PQC schemes, it is a major concern and NIST also in the later rounds encouraged more research on the security of PQC schemes against side-channel cryptanalysis. In relation to this, there has been great research interest in developing new side-channel attacks on all relevant NIST candidates as well as studying efficient side-channel protection techniques.

There are many different approaches to SCA on PQC schemes. Following previous work, we may roughly classify attacks into two main categories. The first category includes attacks that require either a single trace or at least only few traces to perform key recovery or message recovery and targets very precise leakages in an implementation. The second category includes attacks of a more generic type, exploiting arbitrary leakages in the implementation of the algorithm, but typically requiring the collection of many traces in the attack phase. These more generic attacks are modeled by instantiating a side-channel oracle for chosen ciphertexts. The oracle is explained in more detail in the following.

### 1.1 Related works

Key-recovery chosen-ciphertext side-channel attacks (KR-CCA-SCA) are attacks where the adversary recovers the secret key in the scheme by using chosen ciphertext calls to the decryption or decapsulation algorithm and getting measurement data from some side-channel.

KR-CCA-SCA attacks on PQC encryption schemes are a well-established research field, as evidenced by numerous publications [10, 35, 29, 45, 18, 16, 21, 17, 39, 15, 33, 41]. These attacks can be classified depending on where the information leakages are detected. The first type of KR-CCA-SCAs [18, 16, 35] exploits leakages from the two added procedures, the re-encryption and ciphertext comparison, of the FO transform, since these two components in the FO transform depend on the decrypted message vector. There are also KR-CCA-SCAs [29, 21, 17, 39, 15] that exploit side-channel leakages from the CPA-secure decryption, where parts of the decryption procedure will directly use the secret key.

In [34], Ravi et al. classified side-channel-assisted CCA attacks on lattice-based KEMs into three main categories, plaintext-checking (PC) oracle based attacks [10, 35], decryption-failure (DF) oracle based attacks [18], and full-domain (FD) oracle based attacks [29, 45]. The classification depends on what kind of answer the oracle gives. In a DF oracle, the oracle answer is simply whether the chosen ciphertext decodes/decrypts to a valid message or not. On the other hand, a PC oracle and an FD oracle require message recovery before key recovery can take place. In a PC oracle, the response of the oracle is whether the chosen ciphertext results in a specific given message upon decryption. In an FD Oracle, the oracle returns the full message that has been decrypted. As a result, in a PC oracle based attack, it is possible to recover a maximum of one bit of secret information from a single side-channel measurement; however, if the message is of m-bit length (where $m$ is 256 for Kyber), it is possible to recover m bits of secret information with a FD oracle based attack.

Recently, Tanaka et al. in [43] and Rajendran et al. in [32] have independently proposed a new type of oracle called multi-values PC oracle. This oracle can extract 8-12 bits of information from a single decapsulation oracle call through multi-class classification. The multi-values PC oracle can be considered as a compromise between the PC oracle and the FD oracle, although it is still much less efficient than the latter.

For general PQC schemes, Ueno et al. [44] have shown that all round-3 NIST KEM candidates except for Classic McEliece are vulnerable to KR-CCA-SCAs. However, it was later established in [39] that the attack detailed in [44] is only applicable to earlier versions of the HQC proposal and not to the recent Reed-Muller-Reed-Solomon (RMRS) version. Schamberger et al. in [39] and Goy et al. in [15] very recently proposed new power side-channel attacks on the RMRS version of the HQC scheme, but their attack only applied to power analysis with leakages from the CPA decryption. In [16] a generic PC oracle based attack on the RMRS version of the HQC scheme has been proposed, presented in the format of timing attacks.

One central problem in KR-CCA-SCAs is identifying a generic approach to optimize the selection of chosen ciphertexts, in order to efficiently extract information from side-channel measurements. The main obstacles arise from two primary sources: (1) the inaccuracies that may occur in the construction of oracles, particularly with the most powerful FD oracles, and (2) the non-uniform distribution from which secret symbols are generated. To overcome these chal-

lenges, it is necessary to incorporate concepts from coding theory, particularly in the areas of source coding and error correction. Several early research efforts are made to address these challenges, as documented in [29, 31, 41]. But the existing solutions are limited in scope, either because they are restricted to a specific oracle or because they are applicable only to particular types of side-channel leakages. Finally, there is ample room for improvement in terms of attack efficiency.

### 1.2 Contributions

In this paper, we propose a framework named SCA-LDPC to improve the key-recovery side-channel attacks on CCA-secure PQC encryption schemes. The basic idea is to construct chosen ciphertext queries to an oracle that collects information on a set of secret variables in a single query. Then a large number of such queries are considered, each related to a different set of secret variables, and they are modeled as a low-density parity-check code (LDPC code). The secret variables are then determined through efficient iterative decoding methods, such as belief propagation (BP), using soft information.

*New concepts.* The concept of designing chosen-ciphertexts to gather side-channel information in a linear parity check is a fresh and innovative approach. This approach has the potential to provide both source compression and error correction simultaneously. The reason for this is that the combination of multiple secret entries is typically more closely aligned with the uniform distribution, which allows for more effective extraction of information from a single side-channel measurement. This source compression gain can result in a substantial improvement for HQC where secret symbols have an extremely low entropy, as well as a noticeable improvement for lattice-based schemes. The error correction gain is realized through the utilization of linear parity checks, which enable the utilization of correctly recovered coefficients to rectify incorrect decisions. The implementation of these linear checks in the form of an LDPC code was selected due to its efficient decoding capabilities and its well-known near-optimal performance from an information-theoretical perspective.

The new framework has significantly transformed the design philosophy of prior methods for source compression and error correction, as documented in [29, 31, 41]. The previously proposed methods aimed to achieve full key recovery with higher accuracy by introducing additional measurements for each individual secret symbol, thereby increasing the success rate of symbol recovery. The new framework, however, proposes a novel approach by allowing for fewer measurements on the secret symbols, leading to a higher level of symbol-level errors, which are subsequently corrected by the specially designed LDPC codes through inter-symbol parity checks.

We emphasize that the framework is generic in nature and can be applied to both code-based and lattice-based schemes, across adaptive and non-adaptive attack models, and in a multitude of side-channel leakage scenarios, including timing, cache-timing, power, and electromagnetic leakages. To demonstrate the

applicability of the framework, we have instantiated it in two relevant applications: an adaptive timing attack on an HQC implementation with PC oracles, and a non-adaptive power attack on a Kyber implementation with FD oracles. The choice of Kyber and HQC as the primary targets was motivated by their significance, with Kyber being selected as the primary KEM/PKE algorithm for standardization by NIST and HQC still being considered for standardization at the end of round-4.

*New results.* We list the contributions of the paper in the following.

- We introduce a code design method in designing capacity-approaching LDPC /QC-LDPC codes over binary and non-binary alphabets. Our method establishes a relationship between oracle calls and parity checks in the LDPC code, leading to substantial improvements over previous methods in both noiseless and noisy real-world scenarios. The prior improvement is primarily attributed to source compression, while the latter improvement is the result of a combination of source compression and error correction.
- We simulate the performance with different noise levels and characterize the performance of the new approach through a simulation method. The simulated gains are substantial. For example, when the oracle accuracy is 100%, as is the case for the key misuse oracle or an oracle constructed from highly reliable side-channels such as cache-timing leaks on an Intel-SGX platform [24], we can recover the secret key of hqc-128 with approximately 9,000 traces in the PC oracle. Using the same oracle setting as the ideal oracle in [16], we have achieved an improvement factor of 86.6, as we only need about 10,000 traces, compared to the 866,000 traces reported in [16]. This significant improvement is due to the fact that the HQC secret entries are sampled from a distribution with extremely low entropy and the previously known methods (e.g., in [16]) ignore the potential source compression gain. In the scenario of perfect FD oracles, the number of traces required to recover Kyber-768 is only 7, which meets the Shannon lower bound.
- We perform actual attacks on two target algorithms, Kyber and HQC, in real-world scenarios. The results of our study demonstrate a close alignment, or even an improvement, of the real attack performance when compared to the simulation results. The first attack is a full power analysis on a masked implementation of Kyber-768. The attack was carried out using the ChipWhisperer framework on the open-source `mkm4` library in [22] with the profiling and attack phases performed on two distinct boards, both equipped with ARM-Cortex-M4 CPUs. In the real-world scenario, we obtained FD oracles with varying accuracy levels based on their positions. The average accuracy was estimated to be approximately 95%. The full secret key was successfully recovered with an average of 12 traces. In comparison, the simulation required roughly 17 traces for the same oracle accuracy. The better performance in the real-world scenario can be attributed to the availability of soft information and the possibility of some secret symbols having a high accuracy since they are related to a high-accuracy oracle, which in turn helps

in the correct decoding of other positions through the parity checks of the specially designed LDPC codes.
The second attack is a full timing attack simulation on HQC, validated with a real-world timing oracle. The real-life attack performance highly depends on the targeted platform. On our laptop with an Intel Core i5 CPU, we can achieve full key recovery against hqc-128 with $2^{18}$ decapsulation calls.
– The software for attack and simulation will be made open-source.[3]

It is important to note that, in accordance with previous research, our approach focuses on recovering the entire secret vector through side-channel leakages. The sample complexity can be reduced by performing additional post-processing procedures, such as information set decoding and lattice reduction [8], to recover a portion of the secret entries. The specific reduction in the number of traces depends on the permissible amount of computation for post-processing.

*Comparison with previous studies in [29, 31, 41].* In [31], Qin et al. presented an efficient PC oracle based attack on lattice-based schemes by adaptively choosing a new ciphertext for decryption based on the side-channel information obtained from previous power/electromagnetic measurements. Their approach is similar to the well-known Huffman coding method and can result in a source compression gain. Shen et al. in [41] further extended this work by proposing a detection coding method to identify incorrectly recovered positions and to send additional measurements for those secret positions. Note that these studies are limited to PC oracle based attacks on lattice-based schemes and operate in adaptive mode, resulting in a more restrictive attack model and lower efficiency compared to other attacks based on more powerful oracles. For example, when the oracle accuracy is 95%, it was reported in [41] that 3874 traces are required to attack the Kyber-512 scheme; in contrast, the new attack based on the FD oracle presented in this paper only requires 12 traces in a real attack (or 17 traces in simulation) to attack the Kyber-768 scheme. Furthermore, from a viewpoint of information theory, LDPC codes are attractive due to their near-optimal performance. As a result, they are expected to provide improved performance in scenarios where the oracle accuracy is low, compared to the detection codes proposed in [41].

A relevant study [29] has proposed the extended Hamming coding method to enhance the FD oracle based power attack on the masked Saber, a round-3 NIST PQ KEM candidate, in the non-adaptive attack model. However, this approach does not provide any source coding gain and its error correction is limited to an inner-symbol style, resulting in a lack of inter-symbol connections and a less potent error correction mechanism. A more detailed comparison of our work with [29] can be found in Section 6.3.

### 1.3 Relations to very recent work

The first draft of the work was finalized in October 2022. In December 2022, two relevant studies [5, 11] on the topic of FD-oracle based attacks on masked

[3] `https://github.com/atneit/SCA-LDPC`

implementations of Kyber were made available on the IACR ePrint website. These works extend the applications of the method proposed in [29], and as such, differ from the direction taken by our framework in its design towards a more efficient information extraction. These two works offer no source compression gains and the error correction approach is similar to that proposed in [29]. For example, in [5], the authors constructed a masked and shuffled implementation based on the first-order masked implementation in [22] and reported a minimum required number of traces of 38016. While our reported number is 12, a direct comparison between these two attack instances is invalid as the targeted implementations differ. In [11], Dubrova et al. presented a high probability of success in the recovery of message bits from a masked implementation of Kyber up to the fifth order. Their results suggest that our SCA-LDPC attack framework may perform effectively even against masked implementations with a much higher order. Additionally, the authors identified stronger leakage points from the function `masked_poly_frommsg` compared to `masked_poly_tomsg` found in the initial version of our paper. As a result, we have revised our approach to utilize the new leakage points reported in [11] for a more efficient attack.

In January 2023, Huang et al. presented the first KR-CCA-SCA in the context of cache timing attacks in [24]. Their results include a novel approach for using a PC oracle for the KR-CCA-SCA attack on the RMRS version of the HQC scheme. The accuracy of the PC oracle in the cache scenario is high, with a reported rate of almost 100%. However, their method does not offer any source compression or error correction gains and requires a significantly higher number of oracle calls compared to our SCA-LDPC framework (i.e., 53857 vs. 9000). The potential to integrate their attack concept with our SCA-LDPC framework for source compression and error correction is an area of future investigation.

### 1.4 Organization

The remaining parts of the paper are organized as follows. In Section 2, we present the necessary background information. In Section 3, we present a general description of the new attacking framework. We apply the new attack ideas towards Kyber and HQC, in Section 4 and Section 5, respectively. Then, we present the extensive computer simulation results and real-world attacks in Section 6. We finally conclude the paper and present future directions in Section 7.

## 2 Preliminaries

We present the necessary background in this section. We first provide the employed notations and terminology in coding theory, followed by a description of the two KEM candidates, Kyber and HQC. Finally, we conclude this section by outlining the threat model.

### 2.1 Notations and coding terminology

*Notation.* For a finite set $\mathcal{I}$, the symbol $\#\{\mathcal{I}\}$ denotes the number of elements in $\mathcal{I}$. Let $\mathbb{F}_q$ be the finite field of size $q$, $\lceil x \rfloor$ the rounding function, and $\mathcal{H}$, $\mathcal{G}$, and $\mathcal{K}$ three cryptographic hash functions. The central binomial distribution $\mathsf{B}_\mu$ outputs $\sum_{i=1}^{\mu}(a_i - b_i)$, where $a_i$ and $b_i$ are independently and uniformly randomly sampled from $\{0,1\}$. The Bernoulli distribution $\mathsf{Ber}_\eta$ defines a random variable from $\{0,1\}$, which is 1 with probability $\eta$ and 0 otherwise. The notation $\mathbf{a} \xleftarrow{\$} \mathsf{U}$ denotes that the entries in $\mathbf{a}$ are randomly sampled from the distribution $\mathsf{U}$, where $\mathbf{a}$ is a vector or polynomial. For a set $\mathcal{I}$, $\mathbf{a} \xleftarrow{\$} \mathcal{I}$ means that the entries in $\mathbf{a}$ are uniformly sampled from the set $\mathcal{I}$ at random. For a vector or polynomial $\mathbf{a}$, $\mathbf{a}[i]$ refers to the coefficient of $\mathbf{a}$ at the index $i$. The Shannon's binary entropy function of a random variable $X$ is defined as $H(X) = -\sum_{x\in\mathcal{X}} \mathbf{Pr}\left[X = x\right] \log_2 \mathbf{Pr}\left[X = x\right]$.

*Linear codes.* The Hamming weight of a vector $\mathbf{x}$ is its number of non-zero elements, denoted by $w_{\mathrm{H}}(\mathbf{x})$. We define an $[n, k, d]_q$ linear code $\mathcal{C}$ as a linear subspace over $\mathbb{F}_q$ of length $n$, dimension $k$, and minimum distance $d$. Here minimum distance is defined as the minimum Hamming weight of its non-zero elements. Since a linear code $\mathcal{C}$ is a subspace, we can define it as the image of a matrix $\mathbf{G}$, called a *generator matrix*. We can also define the code $\mathcal{C}$ as the kernel of a matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$. Here $\mathbf{H}$ is called a parity-check matrix of $\mathcal{C}$.

*LDPC codes. Low-density parity-check* (LDPC) codes are linear codes with a sparse parity-check matrix first introduced in [14]. LDPC codes can be considered sparse graph codes because they can be decoded efficiently using iterative decoding (such as belief propagation [30]) on the *Tanner graph*, a bipartite graph with edges corresponding to non-zero elements in the parity-check matrix $\mathbf{H}$.

*Concatenated codes.* Forney [12] in 1965 firstly proposed the concatenated code construction approach of combining two simple codes called an *inner code* and an *outer code*, respectively, to achieve good error-correcting capability with reasonable decoding complexity. Let the inner code $\mathcal{C}_{in} : \mathcal{A}^k \to \mathcal{A}^n$, the outer code $\mathcal{C}_{out} : \mathcal{B}^K \to \mathcal{B}^N$, and $\#\{\mathcal{B}\} = \#\{\mathcal{A}\}^k$. The concatenated code is a code $\mathcal{C}_{con} : \mathcal{A}^{kK} \to \mathcal{A}^{nN}$. The key of the concatenated code construction method is that the decoding can be done sequentially by passing first the inner code decoder and then the outer code decoder. Typically in the inner code decoding, one can use a maximum-likelihood decoding approach, while the outer code allows efficient decoding in polynomial time (e.g. by employing an LDPC code).

### 2.2 Kyber

Kyber [40], the KEM version of the Cryptographic Suite for Algebraic Lattices (CRYSTALS), is based on the module Learning with Errors (MLWE) problem and has been solicited as the KEM/PKE standard in the NIST PQ project.

Kyber achieves the IND-CCA security through a tweaked Fujisaki-Okamoto transform [13] transforming an IND-CPA-secure PKE KYBER.$\mathsf{CPAPKE}$ to an

Table 1: Parameter sets for Kyber [40]

| | $n_{\mathsf{mod}}$ | $d$ | $q$ | $\mu_1$ | $\mu_2$ | $(d_u, d_v)$ |
|---|---|---|---|---|---|---|
| Kyber-512 | 256 | 2 | 3329 | 3 | 2 | (10,4) |
| Kyber-768 | 256 | 3 | 3329 | 2 | 2 | (10,4) |
| Kyber-1024 | 256 | 4 | 3329 | 2 | 2 | (11,5) |

IND-CCA-secure KEM KYBER.CCAKEM. The description algorithms of KYBER.CPAPKE and KYBER.CCAKEM can be found in [40]. We include a simplified description in Figure A.1 and Figure A.2 for completeness, where the implementation details with the Number Theoretical Transform (NTT) are omitted.

In the following, we define the compression function and the decompression function, i.e., $\mathbf{Comp}_q(x, d)$ and $\mathbf{Decomp}_q(x, d)$, respectively.

**Definition 1.** *The Compression function is defined as:* $\mathbb{Z}_q \to \mathbb{Z}_{2^d}$

$$\mathbf{Comp}_q(x, d) = \left\lceil \frac{2^d}{q} \cdot x \right\rfloor \pmod{2^d}. \tag{1}$$

**Definition 2.** *The Decompression function is defined as:* $\mathbb{Z}_{2^d} \to \mathbb{Z}_q$

$$\mathbf{Decomp}_q(x, d) = \left\lceil \frac{q}{2^d} \cdot x \right\rfloor. \tag{2}$$

The compression and decompression function can be done coefficient-wise if the input is a polynomial or a vector of polynomials $\mathbf{x} \in \mathcal{R}_q^d$. The procedure $\mathsf{KDF}(\cdot)$ denotes a key-derivation function.

The security parameter sets for the three versions of Kyber, Kyber-512, Kyber-768, and Kyber-1024 are shown in Table 1. In Kyber $q$ is a prime 3329. Let $\mathcal{R}_q$ be a polynomial ring $\mathbb{F}_q[x]/(x^{256}+1)$. Let $\mathbf{H}_0$ be a *negacyclic* matrix from a vector $\mathbf{h}_0$, i.e. the first row is $\mathbf{h}_0$, subsequent rows are cyclically shifted, when the value is moved from the last column to the first one, it is multiplied by -1. Let $d$ denote the rank of the module, set to be 2, 3, and 4, respectively, for Kyber-512, Kyber-768, and Kyber-1024. When sampling from central binomial distribution $\mathsf{B}_\mu$, Kyber also has two parameters ($\mu_1$, $\mu_2$), set to be $(3, 2)$ for Kyber-512 and $(2, 2)$ for Kyber-768 and Kyber-1024.

### 2.3 HQC

HQC (Hamming Quasi-Cyclic) [2] is one of the main code-based IND-CCA-secure KEMs in the NIST PQ project, which has advanced to the fourth round. Its security is based on the hardness of decoding a random quasi-cyclic code in the Hamming metric. In HQC, the base field is $\mathbb{F}_2$ and $\mathcal{R}_2$ denotes the polynomial ring $\mathbb{F}_2[x]/(x^n-1)$. The multiplication of two polynomials $\mathbf{u}, \mathbf{v} \in \mathcal{R}_2$ can be

represented as a vector and a *circulant matrix*, induced from a vector in $\mathbb{F}_2^n$. Given $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in \mathbb{F}_2^n$, its corresponding circulant matrix is defined as

$$\mathbf{rot}(\mathbf{y}) = \begin{pmatrix} y_1 & y_n & \cdots & y_2 \\ y_2 & y_1 & \cdots & y_3 \\ \vdots & \vdots & \ddots & \vdots \\ y_n & y_{n-1} & \cdots & y_1 \end{pmatrix}.$$

We can write the multiplication of $\mathbf{uv}$ as $\mathbf{u} \cdot \mathbf{rot}(\mathbf{v})^{\mathrm{T}}$ or $\mathbf{v} \cdot \mathbf{rot}(\mathbf{u})^{\mathrm{T}}$. The transpose of the circulant matrix is the counterpart of the negacyclic matrix.

The detailed description of the IND-CPA-secure PKE version of HQC and the IND-CCA-secure KEM version can be found in the HQC reference document [2]. We also list them in Figure A.3 and Figure A.4 for completeness. The procedure $\mathsf{KeyGen}(\cdot)$ randomly generates two private vectors $\mathbf{x}, \mathbf{y} \in \mathcal{R}_2$ with a low Hamming weight $w$ as the private key. It also generates a random public vector $\mathbf{h} \in \mathcal{R}_2$, computes $\mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$, and returns $(\mathbf{h}, \mathbf{s})$ as the public key. The scheme employs a linear code $\mathcal{C}$ with a generator matrix $\mathbf{G}$ and generates noise $\mathbf{e}, \mathbf{r}_1, \mathbf{r}_2 \in \mathcal{R}_2$ with low Hamming weight in the encryption. The encryption function computes $\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$ and $\mathbf{v} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}$ and returns $(\mathbf{u}, \mathbf{v})$ as the ciphertext. In decryption, the secret vector $\mathbf{y}$ is an input and it computes

$$\mathbf{v} - \mathbf{u} \cdot \mathbf{y} = \mathbf{mG} + \underbrace{\mathbf{s} \cdot \mathbf{r}_2 - \mathbf{u} \cdot \mathbf{y} + \mathbf{e}}_{\hat{\mathbf{e}}}. \tag{3}$$

Since $w_{\mathrm{H}}(\hat{\mathbf{e}})$ is small, the decryption function inputs $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ to the decoder of $\mathcal{C}$ and can succeed with high probability.

The parameter sets of HQC are shown in Table 2. In the recent version published in June 2021, HQC employs a concatenation of outer $[n_1, k_1, n_1 - k_1 + 1]_{256}$ Reed-Solomon (RS) codes and inner duplicated Reed-Muller (RM) codes built from the first-order $[128, 8, 64]_2$ Reed-Muller code. The encoding procedure first encodes a message $\mathbf{m} \in \mathbb{F}_2^{8k_1}$ to a codeword $\hat{\mathbf{m}} \in \mathbb{F}_{2^8}^{n_1}$ of the employed shortened Reed-Solomon codes. It then maps each byte of $\hat{\mathbf{m}}$ to a codeword of the first-order RM and repeats the RM codeword for 3 or 5 times depending on the security level to obtain a duplicated RM codeword in $\mathbb{F}_2^{n_2}$. In summary, we employ a linear code $\mathbf{mG} \in \mathbb{F}_2^{n_1 n_2}$. The HQC proposal makes all computations in the ambient space $\mathbb{F}_2^n$ and truncates the remaining $n - n_1 n_2$ useless bits.

The IND-CCA security of the KEM version of HQC is achieved by the Hofheinz-Hövelmanns-Kiltz (HHK) transform [23].

### 2.4 Threat model

We consider a side-channel-assisted chosen-ciphertext attack on a KEM's decapsulation algorithm, where the attacker selects ciphertexts and observes specific side-channel data, such as timing [18], cache-timing [24], or power/electromagnetic leakages [35], from the targeted device, which can be a high-end CPU, low-end CPU (e.g., ARM cortex-M4), or hardware device.

Table 2: The HQC parameter sets [2]. The inner code is the duplicated Reed-Muller code defined by the first-order $[128, 8, 64]_2$ Reed-Muller code.

| | RS-S | | | Duplicated RM | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instance | $n_1$ | $k_1$ | $d_{\text{RS}}$ | Mult. | $n_2$ | $d_{\text{RM}}$ | $n_1 n_2$ | $n$ | $\omega$ | $\omega_r = \omega_e$ |
| hqc-128 | 46 | 16 | 31 | 3 | 384 | 192 | 17 664 | 17 669 | 66 | 75 |
| hqc-192 | 56 | 24 | 33 | 5 | 640 | 320 | 35 840 | 35 851 | 100 | 114 |
| hqc-256 | 90 | 32 | 49 | 5 | 640 | 320 | 57 600 | 57 637 | 131 | 149 |

Specifically, we assume that a communication party Alice is using her device for key establishment. An adversary called Malory sends selected ciphertexts to Alice to recover Alice's long-term secret key. Alice runs the decapsulation algorithm and Malory will fail if the used KEM algorithm is IND-CCA secure. However, the designed side-channel-assisted CCAs can make the attack successful after a few such attempts, using the observed side-channel leakages.

This side-channel-assisted CCA attack model is well-established – it is stated in [44] that all the NIST round-3 KEM candidates except for Classic McEliece are vulnerable to such attacks exploring leakages from FO transform. The basic idea is to construct a plaintext-checking(PC) oracle outputting whether $\mathbf{Dec}(c') \stackrel{?}{=} m$, where $c'$ is the chosen ciphertext and $m$ is a message vector.

Finally, the attacker recovers the long-term secret keys based on the output of the PC oracle. Since the PC oracle is generally built from measurements of side-channel leakages, it cannot be 100% correct, in practice. We denote the accuracy of the constructed PC oracle $\rho$, i.e., the oracle outputs the right decision with probability $\rho$ and the wrong one with probability $1 - \rho$.

Note that we are discussing general methods for near-optimal CCA SCAs. This new coding-theoretical approach for reduced sample (trace) complexity can be applied in various side-channel attacks on various platforms, while the starting oracle accuracy $\rho$ can be different. A PC oracle with 100% correctness ($\rho = 1$) can also be connected to a key misuse attack model, as described in [31].

*Profiled power/EM attacks.* Specific to power/EM attacks, we mainly consider a profiled setting that the adversary has a similar but different device to perform training activities. Though the adversary has no access to the secret key in the targeted device, the secret key in the training device can be freely set. We can also apply the new idea to non-profiled attacks that can build the required abstract oracles online, but the sample complexity analysis will be different.

*Comparison with the adaptive model in power/EM attacks.* The studies [31, 32, 41] proposed efficient adaptive KR-CCA-SCAs on lattice-based proposals. This attack model allows the adversary to select a new chosen ciphertext based on information obtained from previous power/EM traces, which can be employed for source coding on secret coefficients. This approach can result in reduced sample complexity close to the lower Huffman or Shannon bounds. However, this attack

model is strong for many practical (say IoT) applications since the adversary needs to have good connections with the device measuring the power/EM leakages and good computation capability to instantly process the obtained traces. We highlight that our new SCA-LDPC attack framework eliminates the requirement and offers source coding gain in a non-adaptive attack model.

## 3 General Description of the SCA-LDPC Attack Framework

This section presents a new idea of incorporating LDPC codes and soft information to design chosen ciphertexts and improve previously established KR-CCA-SCAs for CCA-secure post-quantum Key Encapsulation Mechanisms (KEMs) and encryption schemes. We propose a novel technique to extract, from a single side-channel measurement, information regarding a low-weight parity check of the secret coefficients, as opposed to information regarding a single coefficient in previous methods. The sparse system is then solved using iterative decoding methods, such as belief propagation. This new approach enables the attainment of both source compression benefits and error correction advantages. This is due to the combination of several secret coefficients, which leads to a more uniform extraction of information from a single trace. Additionally, the correct recovery of coefficients facilitates the correction of erroneous decisions through spare parity-check relations. The adoption of this new method significantly reduces the number of necessary side-channel measurements. We call the new attack strategy a framework as it is generic and can be applied to both code-based and lattice-based schemes, in a multitude of side-channel leakage scenarios including timing, cache-timing, power, and electromagnetic leakages.

We start this section by assuming the availability of a well-designed LDPC code with specific dimensions and proceed to explain its utilization for improved side-channel information extraction. We then in Section 3.2 present a simple method for constructing such linear codes, the effectiveness of which will be demonstrated through experiments in Section 6. In addition, we broaden the framework by introducing a concatenated construction, where the LDPC codes are utilized as the outer code. This construction is particularly efficient for lattice-based schemes that feature a large alphabet size or for scenarios where the accuracy of the oracle constructed from side-channel measurements is limited.

### 3.1 New attack idea

Given a good linear code with a sparse parity-check matrix $\mathbf{H}_{r\times n}$, there are $k = n - r$ secret positions to recover. In lattice-based and code-based KEM proposals, the value $k$ is usually divided into $b$ blocks, each of which has the size of $k/b$. We add the constraint that $\mathbf{H}$ should have the form of

$$\mathbf{H} = \left[\mathbf{H}_{r\times k} | -\mathbf{I}_{r\times r}\right].$$

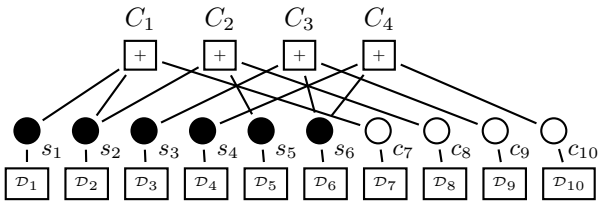


Fig. 1: The Tanner graph explanation.

Our goal is to recover the first $k$ secret entries $s_i$. One parity-check equation, i.e., one row in the parity-check matrix $\mathbf{H}$, will introduce one check variable $c_i$ for $i \in \{k+1, \ldots, k+r\}$. We can rewrite each parity-check equation as $c_i = \sum_{j \in \mathcal{I}} s_j$ and the size of $\#\{\mathcal{I}\}$ is small since the matrix $\mathbf{H}$ is of low density.

The secret entries $s_i$ are typically generated according to a certain secret distribution. For example, in the lattice-based scheme Kyber, the secret entries are generated from the central binomial distribution $\mathsf{B}_\mu$; in the code-based scheme HQC, the secret vector is very sparse and each secret entry can be viewed as a Bernoulli variable $\mathsf{Ber}_\eta$, where $\eta$ is a small positive number. The secret distribution can be utilized as the prior information for $s_i$. Moreover, additional information can be obtained through the implementation of side-channel measurements of $s_i$, which subsequently updates the relevant distribution. This approach is particularly beneficial in lattice-based scenarios. We then design new ciphertexts to obtain side-channel leakages of sparse linear combination $c_i$ of $s_j$ for $j \in \mathcal{I}$. The side channel information could reveal an empirical probability of $c_i$. The problem of recovering all $s_i$ for $i \in \{1, \ldots, k\}$ is transformed into a coding problem through a noisy discrete channel. Note that the design method for ciphertexts that can reveal partial information of $c_i$, is unique to each proposed scheme and differs between lattice-based and code-based schemes. This ciphertext design is one main technical challenge in the proposed attack framework.

*Explanation.* The attack idea is illustrated in Figure 1. We assume that 6 secret coefficients or variables $s_i$ for $1 \leq i \leq 6$ need to be recovered. For each $s_i$, we can use the a priori distribution (e.g., in the HQC case), or we have more traces or oracle calls to get a better knowledge of its distribution (e.g., in the Kyber case). We show 4 parity checks in this example, and each check connects to a new variable $v_i$. From side-channel measurements or oracle calls, we got additional information about these variables. Thus, we could assign the corresponding distribution to these variables and build a Tanner graph as in Figure 1. With this sparse bipartite Tanner graph, we perform iterative decoding to recover the desired secret coefficients $s_i$ for $1 \leq i \leq 6$.

*The gain of using LDPC codes.* It is essential to select a sparse graph code that facilitates efficient decoding and renders the key recovery procedure computationally feasible. Therefore, it is natural to examine LDPC codes that have favor-

able characteristics from an information-theoretic point of view. We introduce the variables $c_i$, which are sparse linear combinations of the secret coefficients $s_j$, thereby facilitating a more efficient extraction of information from a single side-channel measurement. This is due to the fact that the distribution of $c_i$ is typically closer to a uniform distribution compared to the distribution of $s_i$, resulting in substantial source compression gains, particularly in the case of HQC and to a significant extent in Kyber. Further discussions regarding these source compression gains will be presented in Section 6. Finally, LDPC codes can offer close to optimal error correction performance, rendering the attack framework efficient in terms of the number of side-channel measurements required, even when the oracle constructed from side-channel leakages is highly inaccurate.

*Example 1 (The source compression gain for* hqc-128*).* In hqc-128, the length of $\mathbf{y}$ is $n = 17669$ and the Hamming weight of $\mathbf{y}$ is $w_{\mathrm{H}}(\mathbf{y}) = 66$. Hence, we can approximate each position of $\mathbf{y}$ as a Bernoulli distribution $\mathsf{Ber}_\eta$, where $\eta \approx 0.0037$. Assume that we have a perfect oracle to inform us of the value of one position from one oracle call. If we try to recover a bit in $\mathbf{y}$ by one oracle call, with Shannon's binary entropy function, the obtained information is bounded by 0.0352 bit. If we xor 50 i.i.d. secret positions (as we did later in Section 6) and try to recover the new random bit from one oracle call, then we can instead obtain 0.6255 bit of information. Thus, from an information-theoretical perspective, the new framework is much more advantageous.

### 3.2 Code generation

It has been demonstrated in previous research [38] that random sparse linear codes exhibit superior decoding performance and specific classes of Low-Density Parity-Check (LDPC) codes, such as [37], can attain error-correction capabilities that approach the Shannon capacity. In this work, we present a straightforward code construction method that has shown remarkable results in our experiments.

We first borrow the concept of distance spectrum from [19].

**Definition 3 (Distance Spectrum [19]).** *For a binary vector $\mathbf{h} \in \mathbb{F}_2^{n_0}$, we define its distance spectrum $D(\mathbf{h})$ as*

$$D(\mathbf{h}) = \{d : 1 \le d \le \lfloor n_0/2 \rfloor, d \text{ classified as existing in } \mathbf{h}\},$$

*where "existing in $\mathbf{h}$" means there are two ones in $\mathbf{h}$ with distance $d$ or $(n_0 - d)$ inbetween. A distance $d$ can appear many times in the distance spectrum of a given bit pattern $\mathbf{h}$. We call this number the multiplicity of $d$.*

In our new attack, we first generate QC-LDPC codes with $mb$ blocks of the parity-check matrix

$$\mathbf{H}_{\mathsf{ini}} = \begin{bmatrix} \mathbf{H}_{11} & \cdots & \mathbf{H}_{1b} \\ \vdots & \ddots & \vdots \\ \mathbf{H}_{m1} & \cdots & \mathbf{H}_{mb} \end{bmatrix},$$

where $\mathbf{H}_{ij}$ is the circulant matrix (or the negacyclic matrix in the $q$-ary case) generated from a binary vector $\mathbf{h}_{ij}$ for $1 \leq i \leq m, 1 \leq j \leq b$ with a low Hamming weight. We generate the vectors $\mathbf{h}_{ij}$ randomly with the constraint that only distances of multiplicity 1 are allowed in its distance spectrum. This can be done with high probability since the constructed LDPC codes are sparse. The key point in the design is that a length-4 cycle occurs in the associated Tanner graph if the multiplicity of a distance in the distance spectrum is larger than 2. By avoiding such patterns in a block, we can avoid many length-4 cycles; such attempts can improve the decoding performance as length-4 cycles can substantially hurt the decoding performance.

We select $r$ rows of $\mathbf{H}_{\mathsf{ini}}$ (randomly or according to certain rules) to form a sub-matrix $\mathbf{H}'$ and append $-\mathbf{I}_{r\times r}$, where $\mathbf{I}_{r\times r}$ is the identity matrix. Thus, the parity-check matrix of the final generated code is

$$\mathbf{H} = \left[\mathbf{H}'_{r\times n_0 b} | -\mathbf{I}_{r\times r}\right] \tag{4}$$

*Concatenated code construction.* The LDPC codes generated from the above simple approach can serve as the outer code in the concatenated construction. The inner code can be any linear code such as a repetition code, the extended Hamming codes, and a further concatenation of the extended Hamming codes and repetition codes in [29]. Moreover, we can include a soft-input-soft-output decoder (e.g., in [25]) to utilize the soft-information. Note that in the soft-decoding procedure (e.g., the BP algorithm) of the outer code, only a distribution of each secret coefficient random variable is required; we could thus employ a code with an efficient maximum likelihood decoding procedure as the inner code allowing an efficient calculation of the soft output of the coefficient distribution.

In summary, such concatenated code construction enhances decoding capability and also balances decoding complexity, as the decoding of both outer and inner codes is efficient. This construction is particularly effective for lattice-based proposals or when the side-channel oracle exhibits a low level of accuracy.

## 4 Application to Kyber

In this section, we outline the details of how the new SCA-LDPC framework can be applied to Kyber. The attack is more effective for Kyber if we have side-channel leakages for both $s_i$ and $c_j$. We demonstrate how to obtain these leakages, construct inner codes for them, and apply the outer LDPC decoder.

### 4.1 Basic key recovery attack

In the following, we explain the basic attack to obtain side-channel information about secret coefficients $s_i$. We focus on Kyber-768 mostly because the new protected implementation [22] that we target supports only this set of parameters. For Kyber-768, the secret key is $\mathbf{s} = (\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2)$, a ciphertext is a pair $(\mathbf{u}', \mathbf{v}')$. To decrypt a ciphertext, one computes $\mathbf{m} = \mathbf{Comp}_q(\mathbf{v} - \mathbf{s}^{\mathrm{T}}\mathbf{u}, 1)$,

where $\mathbf{u} = \mathbf{Decomp}_q(\mathbf{u}', d_u) = (\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2)$, $\mathbf{v} = \mathbf{Decomp}_q(\mathbf{v}', d_v)$. The common practice [35] is to choose a ciphertext that leads to $\mathbf{m} = (0, 0, \ldots, 0)$ or $\mathbf{m} = (1, 0, \ldots, 0)$. In other words, all bits of the message are fixed to 0 except the first one. This can be done, for example, by setting $\mathbf{u}_0 = (k_u, 0, \ldots, 0)$, $\mathbf{u}_1 = \mathbf{u}_2 = \mathbf{0}$ and $\mathbf{v} = (k_v, 0, \ldots, 0)$, where $k_u, k_v$ are some numbers modulo $q$. In this case, the message bits are subject to the following equation.

$$\mathbf{m}[i] = \begin{cases} \mathbf{Comp}_q(k_v - k_u \cdot \mathbf{s}_0[0], 1), & i = 0 \\ \mathbf{Comp}_q(k_u \cdot \mathbf{s}_0[i], 1), & i \geq 1 \end{cases} \tag{5}$$

By choosing appropriate values for $k_u$ and $k_v$, it is possible to force $\mathbf{m}[i]$ to always be zero for $i \geq 1$, while the value of $\mathbf{m}[0]$ depends on the first secret coefficient $\mathbf{s}_0[0]$. Since secret coefficients for Kyber-768 are taken from the range $[-2, \ldots, 2]$, some of the coefficients are encoded as 0, while others are encoded as 1. We can use several such ciphertexts with (possibly) different $k_v$ and/or $k_u$ to get an inner code of longer length. This way, using an oracle that distinguishes message $(1, 0, \ldots, 0)$ from $(0, 0, \ldots, 0)$, the attacker can get the distribution of a secret coefficient closer to the real value the more ciphertexts he uses.

There are restrictions for the values $k_u$ and $k_v$: (1) these values are taken from the image of $\mathbf{Decomp}_q$; (2) $k_u$ is chosen in a way such that $\mathbf{Comp}_q(k_u \cdot s, 1) = 0$ for any secret coefficient $s$ (follows from Equation (5)). Thus, one cannot use any code; even though it is possible to encode each secret coefficient with only $\lceil \log_2(5) \rceil = 3$ bits, for any fixed in-advance combination of 3 ciphertexts one cannot fully determine an arbitrary secret coefficient even with perfect oracle.

One way to solve this problem [41] is to choose ciphertexts adaptively based on the output of the oracle, but we take a different approach. Consider an FD oracle based attack, i.e., assume that we have a set of oracles $(\mathcal{O}_i)_{i \in (0..n-1)}$, where $n$ is the length of the message. Given a ciphertext, the oracle $\mathcal{O}_i$ says if $\mathbf{m}[i] = 1$ or not. Essentially, the attacker calls all of these oracles at once, giving them the same ciphertext, this way he can get information about the whole message to be decrypted, the scenario is the same as in [29]. The attacker can create a ciphertext in the following way, set $\mathbf{u}_0 = (k_u, 0, \ldots, 0)$, $\mathbf{u}_1 = \mathbf{u}_2 = \mathbf{0}$, and $\mathbf{v} = (k_v, k_v, \ldots, k_v)$, then

$$\mathbf{m}[i] = \mathbf{Comp}_q(k_v - k_u \cdot \mathbf{s}_0[i], 1), \tag{6}$$

i.e., $i^{\text{th}}$ bit of the message depends on $\mathbf{s}_0[i]$. Thus, from one ciphertext the information about the block of 256 coefficients $\mathbf{s}_0$ can be obtained. Since there is no more restriction on $\mathbf{m}[i] = 0$ for $i \geq 1$, the amount of possible inner codes increases greatly. Table 3 shows an inner code from three ciphertexts built from $(k'_u, k'_v)$ pairs, this code can be used to fully determine 256 secret coefficients with perfect oracles. Note that to create an actual ciphertext $(\mathbf{u}', \mathbf{v}')$ we need a pair $(k'_u, k'_v)$ that maps to $(k_u, k_v)$ with coefficient-wise function $\mathbf{Decomp}_q$. The next block of secret coefficients $\mathbf{s}_1$ can be retrieved by setting $\mathbf{u}_0 = \mathbf{0}$, $\mathbf{u}_1 = (k_u, 0, \ldots, 0)$, $\mathbf{u}_2 = \mathbf{0}$ and so on. Note that the attacker could choose different values in $\mathbf{v}$, this way different encodings can be used for different message

Table 3: Example of an inner code for the secret coefficients. Each value from the range $[-2, \ldots, 2]$ is encoded with 3 bits (columns of the table), therefore, the secret coefficient could be fully determined with just 3 oracle calls given that the oracle is perfect.

| $(k'_u, k'_v)$ | Secret coefficient | | | | |
|---|---|---|---|---|---|
| | -2 | -1 | 0 | 1 | 2 |
| $(630, 14)$ | 0 | 1 | 0 | 1 | 1 |
| $(706, 6)$ | 0 | 0 | 1 | 1 | 0 |
| $(706, 10)$ | 0 | 1 | 1 | 0 | 0 |

bits (although all those encodings should have the same $k_u$) and this potentially opens up the possibility of the adaptive attack. However, such an attack is more complicated since the set of allowed encodings given the fixed $k_u$ is quite limited, and the attacker has to choose the same $k_u$ for all 256 coefficients. We leave it as a potential follow-up work and focus on the situation where for all message bits there is a fixed in-advance encoding to be used.

The common approach in the literature is to use just an inner code for secret coefficients (without outer code) that makes the probability of getting the wrong coefficient to be very small (with real imperfect oracles), such that the probability to get all secret coefficients correctly is close to 1. In our approach, however, we use a shorter inner code that is not sufficient by itself, for example in our real attack from Section 6.1 we encode each secret coefficient with only 2 bits and encode the values $-2$ and 2 the same way, i.e., with only inner code it is impossible to differentiate between these values.

*How to choose inner code.* For the fixed in-advance code length $\ell$ we want to create an inner code $C_\ell$ that maximizes the information we get from the oracles with accuracy $\rho$. We solve this problem by considering the entropy of secret coefficients. Initially, each of them is distributed according to $\mathsf{B}_\mu$, whose entropy is $H(\mathsf{B}_\mu) \approx 2.03$, for $\mu = 2$. Each value $s \in \mathsf{B}_\mu$ is encoded as $C_\ell(s)$ – a binary string of length $\ell$. Given an output string $\mathbf{y}$ of length $\ell$ from an oracle (note that $\mathbf{y}$ can be different from every $C_\ell(s)$, $s \in \mathsf{B}_\mu$), consider the probability $\mathbf{Pr}[\mathsf{B}_\mu = s \mid \mathbf{y}]$ for each $s \in \mathsf{B}_\mu$. As an example from Table 3, $\mathbf{Pr}[\mathsf{B}_\mu = 0 \mid 011] = 1$ for the perfect oracle, but it is less than 1 for an oracle with $\rho < 1$ since we could have reached this $\mathbf{y}$ from another coefficient.

To avoid ambiguity, we denote $\mathbf{y}_\rho$ as the output of the oracle with the accuracy $\rho$. The conditional distribution $\mathsf{B}_\mu|\mathbf{y}_\rho$ can be naturally defined as $\mathbf{Pr}[(\mathsf{B}_\mu|\mathbf{y}_\rho) = s] = \mathbf{Pr}[\mathsf{B}_\mu = s|\mathbf{y}_\rho]$. Now, the difference between the entropy values $H(\mathsf{B}_\mu) - H(\mathsf{B}_\mu|\mathbf{y}_\rho)$ shows how much information the output $\mathbf{y}_\rho$ gives. To assess how good the code is, we can compute the expectation of this information as

$$I(C_\ell) = \sum_{\mathbf{y}_\rho \in \{0,1\}^\ell} (H(\mathsf{B}_\mu) - H(\mathsf{B}_\mu|\mathbf{y}_\rho)) \cdot \mathbf{Pr}[Y = \mathbf{y}_\rho],$$

where $Y$ is a random variable that describes the output of an oracle with accuracy $\rho$ on a random secret coefficient. The probability of the specific oracle's output is computed as follows.

$$\mathbf{Pr}\left[Y = \mathbf{y}_\rho\right] = \sum_{x \in \mathrm{supp}(\mathsf{B}_\mu)} \rho^{d(\mathbf{y}_\rho, C_\ell(x))}(1-\rho)^{\ell - d(\mathbf{y}_\rho, C_\ell(x))}\mathbf{Pr}\left[\mathsf{B}_\mu = x\right],$$

where $d(\cdot,\cdot)$ is the Hamming distance. To decode a received word $\mathbf{y}_\rho$, one computes conditional probability $\mathsf{B}_\mu|\mathbf{y}_\rho$ of secret coefficient, i.e. we use maximum-likelihood decoding approach.

### 4.2 Improving the attack using LDPC

The basic attack allows us to compute the conditional distribution for each secret coefficient using the inner code. Now, following our framework, we create an outer LDPC code. For it to work, we also need a way to get information about parity checks $c_i$. Let us describe how to create a ciphertext corresponding to a parity check. Consider an example: Let $\mathbf{u}_1, \mathbf{u}_2$ and $\mathbf{v}$ be as above, but $\mathbf{u}_0 = k_u + k_u x^2$, then

$$\mathbf{s}^{\mathrm{T}}\mathbf{u} = k_u\left((\mathbf{s}_0[0] - \mathbf{s}_0[n-2]) + (\mathbf{s}_0[1] - \mathbf{s}_0[n-1])x + (\mathbf{s}_0[2] + \mathbf{s}_0[0])x^2 + \ldots\right).$$

Looking at the first message bit

$$\mathbf{m}[0] = \mathbf{Comp}_q(k_v - k_u \cdot (\mathbf{s}_0[0] - \mathbf{s}_0[n-2]), 1)$$

and comparing it to Equation (6), one can recover $c_0 \leftarrow \mathbf{s}_0[0] - \mathbf{s}_0[n-2]$ using a similar approach as in recovering $\mathbf{s}_0[0]$ with $\mathcal{O}_0$. However, $c_0$ lies in the range $[-4, \ldots, 4]$, therefore the recovery process is more complicated. However, we still use several different ciphertexts to get an inner code for the check variables. In other words, there are two inner codes: one for secret coefficients, and another one for check variables. Each of them helps us to compute conditional distributions, which we use with outer LDPC code.

Now, let us represent $c_0$ as a vector $\mathbf{h}_0$ with values from $\{-1, 0, 1\}$ such that $c_0 = \mathbf{h}_0^{\mathrm{T}}(\mathbf{s}_0[0], \ldots, \mathbf{s}_0[n-1])$. In general, if $\mathbf{u}_0 = k_u \cdot \sum_{j=1}^{w} x^{i_j}$, then $\mathbf{h}_0$ is a vector with $w$ nonzero entries at the positions $(-i_j) \bmod n$, where the entry is 1 if and only if $i_j = 0$. Let $\mathbf{H}_0$ be a negacyclic matrix of the vector $\mathbf{h}_0$. With this ciphertext, the $i^{\text{th}}$ message bit is connected to the $i^{\text{th}}$ row of $\mathbf{H}_0(\mathbf{s}_0[0], \ldots, \mathbf{s}_0[n-1])^{\mathrm{T}}$. Note that, unlike in Section 3.1, $c_i$ is the sum of secret coefficients, possibly multiplied by -1. However, this does not significantly affect the result since from the distribution of the coefficient it is trivial to obtain the distribution of the negative coefficient and vice versa. Thus, we still call $c_i$ the sum of secret coefficients.

Let $\mathbf{u}_r = k_u \cdot \sum_{j=1}^{w} x^{i_j^{(r)}}$, $r \in \{0, 1, 2\}$. Ciphertext $(\mathbf{u}, \mathbf{v})$ with the help of oracles $\mathcal{O}_i$ reveals information about 256 parity checks. The parity-check matrix of the outer LDPC code in this case is of the form

$$\mathbf{H}_{\mathsf{ini}} = \left[\mathbf{H}_0 | \mathbf{H}_1 | \mathbf{H}_2\right],$$

where $\mathbf{H}_j$ is the negacyclic matrix obtained from the vector connecting $c_0$ and $\mathbf{s}_j$. Due to the FD oracle, parity checks $c_1, \ldots, c_{n-1}$ must be negacyclic shifts of $c_0$. We only demonstrated the parity-check matrix for the outer LDPC code consisting of block of 256 checks, but there could be several such blocks. Note that in general, the polynomials $\mathbf{u}_r$ do not have to use the same $w$.

There are three main ways to increase the success probability of the attack.

1. Increase the length of the inner code for the secret coefficients. Querying oracles as in Section 4.1 leads to a more accurate distribution for each coefficient.
2. Similarly, increase the length of the inner code for the check variables, i.e., fix the indexes $i_j^{(r)}$ and use different $(k_u, k_v)$.
3. Increase the number of check blocks. The resulting parity-check matrix of the LDPC code $\mathbf{H}_{\mathsf{ini}}$ consists of $3 \times m$ blocks of negacyclic matrices, where $m$ is the number of "unique" parity checks $c_0, c_n, c_{2n}, \ldots$

Creating the best inner code for the check variables that maximizes the amount of information is a challenging task. An educated guess would be the most accurate way to describe our approach to tackling this problem.

## 5 Application to HQC

In this section, we describe the detailed attack on HQC. $\mathcal{O}_{\mathrm{HQC}}$ denotes a general side-channel-based PC oracle for HQC, referenced prior-art assumes timing leakage, but this is not required. We treat a key-misuse oracle as a chosen-ciphertext side-channel oracle with 100% oracle accuracy.

### 5.1 Key-recovery attack with $\mathcal{O}_{\mathbf{HQC}}$

In [16] the authors presented a plaintext checking (PC) oracle based on timing information due to the use of rejection sampling. In this section, we describe how the PC attack works and then explain how we can improve it by using our new SCA-LDPC framework, which is based on coding theory.

Currently, HQC makes use of so-called rejection sampling in the CPA secure encryption function [2, 16]. The rejection sampling algorithm is used to construct random vectors with a specific Hamming weight $\omega$. It works by random sampling of bit positions in the vector, and if some positions are sampled twice, they are rejected. Straight-forwardly implemented, this algorithm leaks timing information due to the inherently random number of rejections that occur. The HQC implementations tested in [16] leak timing information mainly through the use of so-called "seedexpander" calls. The output of the seedexpander function is deterministic pseudo-randomness given by an eXtendable Output Function (XOF). The rejection sampling algorithm uses the seedexpander function to generate relatively large blocks of randomness, at a time. The timing distribution, therefore, is highly dependent on the number of seedexpander calls needed. The minimum number of seedexpander calls occurs when there are no rejections in the rejection

sampling algorithm. In practice, we classify timing measurements based on the number of *additional* seedexpander calls. They are each related to one of the four[4] distributions $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$, listed in increasing order of rarity.

Prior to the publication of the referenced work, it was believed that this randomness was only dependent on values known to the attacker, in this case, the plaintext $\mathbf{m}$. The assumption then was that constant time implementation was not needed for the rejection sampling algorithm. Certainly, it was shown in [16] that this assumption is problematic. Although $\mathbf{m}$ is indeed known to the attacker, the result of the implicitly carried out comparison $\mathbf{m}' \stackrel{?}{=} \mathbf{m}$ is not. Here $\mathbf{m}' = \text{decode}(\mathbf{c} + \mathbf{e}')$ and $\mathbf{e}'$ is a extra noise supplied by the attacker.

The authors showed a key-recovery attack where, by using the timing information due to rejection sampling, knowledge of $\mathbf{m}' \stackrel{?}{=} \mathbf{m}$ is leaked. The attack required 866,000 so-called "idealized oracle" ($\mathcal{O}^{\text{ideal}}_{\text{HQC}}$) queries for the 128-bit security setting. The idealized oracle assumes a noise-free environment where a single timing measurement is sufficient to determine the membership of $\mathcal{S}_j$ (where $j = 3$ in [16]). Unfortunately, this is not sufficient for a 100% correct oracle, due to reasons explained in the following paragraph.

*What follows is a high-level summary of the referenced attack;* A plaintext $\mathbf{m}$ is selected according to some criteria useful for the distinguisher. In the case of timing leakage, the distinguishing property is such that the selected $\mathbf{m}$ results in the timing distribution $\mathcal{S}_3$, since it is the one most easily distinguished. The probability of for any random $\mathbf{m}'$, where $\mathbf{m}' \neq \mathbf{m}$, resulting in the same $\mathcal{S}_3$ timing distribution is low (0.58% per [16]). In other words, $\mathbf{m}' \stackrel{?}{=} \mathbf{m}$ can be distinguished with a high, yet-not-complete, advantage.

A ciphertext $\mathbf{c}' = (\mathbf{u}, \mathbf{v})$ is crafted in the next step such that $\mathbf{r_1}$ is $\mathbf{1} \in \mathcal{R}$ and $\mathbf{r_2}$ and $\mathbf{e}$ is $\mathbf{0} \in \mathcal{R}$. By Equation (3) this results in

$$\mathbf{v} - \mathbf{u} \cdot \mathbf{y} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r_2} + \mathbf{e} - (\mathbf{r_1} + \mathbf{h} \cdot \mathbf{r_2}) \cdot \mathbf{y} = \mathbf{mG} - \mathbf{r_1} \cdot \mathbf{y} = \mathbf{mG} - \mathbf{y} \quad (7)$$

which makes $\mathbf{y}$ the only remaining error for the decoder to correct. Note too that by knowledge of $-\mathbf{y} = \mathbf{y}$ it is a simple computation to find the rest of the private key, since $\mathbf{x} = \mathbf{s} - \mathbf{h} \cdot \mathbf{y}$. Calculating $\mathbf{x}$ is quite unnecessary, however, since it is not used in decapsulation.

Plainly, this crafted ciphertext is invalid and will be rejected in the ciphertext comparison step of the decapsulation. However, a valid ciphertext is not required due to the timing leakage in the XOF via the non-constant time rejection sampling algorithm. The reencryption step immediately preceding the comparison derives the values of $\mathbf{r_1}$, $\mathbf{r_2}$ and $\mathbf{e}$ from the XOF seeded by $\mathbf{m}$. The single bit information $\mathbf{m}' \stackrel{?}{=} \mathbf{m}$ leaks prior to the ciphertext comparison step.

Hall et al. proposed in [20] a way to recover $\mathbf{y}$; An additional error vector $\mathbf{e}'$ is added to $\mathbf{c}'$. $\mathbf{e}'$ is of just sufficient weight to cause a decoding failure (i.e. $\mathbf{m}' \neq \mathbf{m}$ is leaked). The basic attack then simply iterates through each bit $0 \leq i \leq N$

[4] Strictly, there is no upper bound, but the practical benefit of finding a value for $\mathcal{S}_{\geq 4}$ is not worth the exponential effort required [16].

of $\mathbf{e}'$ not already flipped to find those positions that if flipped would result in a decoding success. If this is the case for any value of $i$ this indicates that the bit was already flipped in $\mathbf{y}$ in the ciphertext.

However, this technique alone is not sufficient to provide decisions on all bits in the ciphertext. The reason is twofold. First, unflipping a bit in the error pattern given to the RMRS decoder does not guarantee a decoding success, and secondly due to the possibility that both $\mathbf{m}'$ and $\mathbf{m}$ result in the timing distribution $\mathcal{S}_3$, even though $\mathbf{m}' \neq \mathbf{m}$. This is modeled by $\mathcal{O}_{\text{HQC}}^{\text{ideal}}$, the idealized oracle from [16], which though noise-free, is not 100% correct.

The first problem is solved by using many different error patterns $\mathbf{e}'$. The second was solved by majority voting, i.e. by gathering three or more decisions for every bit. Both of these solutions drive up the number of required oracle calls, even in the ideal timing leakage setting. For the 128-bit security level, this number adds up to 866,000 oracle calls [16].

### 5.2 New improved attack using LDPC codes.

What follows is a description of the new attack listed in Figure 2, a PC oracle $\mathcal{O}_{\text{HQC}}$ is assumed. Like in the original attack [16] we select a plaintext with good side-channel detection properties (in the original case this is a timing property).

The next step is to construct a $N \times N$ regular cyclic LDPC parity-check matrix $\mathbf{H}_{\text{ini}}$ without cycles of length 4, with a good decoding performance. $\mathbf{H}_{\text{ini}}$ has a row-weight of $W$. This construction is detailed in Section 3.2, with $(m = 1, b = 1)$. The first row of $\mathbf{H}_{\text{ini}}$ is the vector $\mathbf{h}_{\text{ini}}$.

We craft a special ciphertext $\mathbf{c}'$ where $\mathbf{r_2} = \mathbf{0}, \mathbf{e} = \mathbf{0}$ and $\mathbf{r_1} = \mathbf{h}_{\text{ini}}$. Similarly to the case given by Equation (7) above, this results in

$$\mathbf{v} - \mathbf{u} \cdot \mathbf{y} = \ldots = \mathbf{mG} - \mathbf{r_1} \cdot \mathbf{y} = \mathbf{mG} - \mathbf{h}_{\text{ini}}\mathbf{y} \tag{8}$$

which makes the added noise that the decoder has to correct equal to $\mathbf{h}_{\text{ini}}\mathbf{y}$. In other words, each bit position $i$ in $\mathbf{c}'$ correspond to the result of a parity-check equation over $\mathbf{y}$, given by $\mathbf{h}_{\text{ini}} >> i$ (cyclic shift by $i$ steps) due to the cyclic nature of our LDPC code.

The Reed-Muller (RM) and Reed-Solomon (RS) concatenated (RMRS) decoder, used in HQC, can be attacked in two stages. First we select $(d_{RS} - 1)/2$ outer RM blocks (each RM block decodes to one RS symbol) to flip in $\mathbf{c}'$ (by XOR with $\mathbf{e}'$). This results in a state where if one more block is flipped it will result in a decoding error in the RS decoder. A decoding failure such as that would be detected by $\mathcal{O}_{\text{HQC}}$. We randomly select another block which we denote $B$.

The next stage is to find which bits $\mathcal{I}_{\mathbf{e}'}^{B}$ to flip in the block $B$ that results in a decoding failure. We do this by flipping bits $i \in \mathcal{I}_{\mathbf{e}'}^{B}$ such that $\mathbf{e}'[i] = 1$ in block $B$ until a RM decoding failure occurs. This propagates as a failure symbol to the RS decoder which is already on the brink of being overwhelmed. This results in a state where $\mathbf{c}' + \mathbf{e}'$ fails to decode due to too much additional noise in the block $B$ partition of $\mathbf{e}'$.

**Input:** $\mathcal{O}_{\text{HQC}}$, public key
**Output:** y

Select plaintext $\mathbf{m}$ ▷ *With good side channel distinguishing properties*
Generate sparse vector $\mathbf{h}_{\text{ini}}$ ▷ *According to Section 3.2*
Construct $\mathbf{H}_{\text{ini}}$ ▷ *From $\mathbf{h}_{ini}$ by cyclic shifts*
Craft $\mathbf{c}'$ with $\mathbf{r_2} = \mathbf{0}, \mathbf{e} = \mathbf{0}$ and $\mathbf{r_1} = \mathbf{h}_{\text{ini}}$
$\mu \leftarrow 0^N$ ▷ *Initialize message*
**loop**
    $\mathbf{e}' \leftarrow 0^N$,
    $\mathcal{B}' \leftarrow$ random subset of size $(d_{RS}-1)/2$ from $\{0, \ldots, n_1 - 1\}$
    **for each** $B' \in \mathcal{B}'$ **do** ▷ *Flip $(d_{RS}-1)/2$ RM-blocks*
        Flip block $B'$ in $\mathbf{e}'$
    **end for**
    $B \xleftarrow{\$} \{0, \ldots, n_1\} \setminus \mathcal{B}'$ ▷ *Select a random unflipped block*
    $\mathcal{I}^B \leftarrow \{Bn_2, \ldots, B(n_2+1) - 1\}$
    $\mathcal{I}^B_{\mathbf{e}'}, \mathcal{I}^B_0, \mathcal{I}^B_1, \leftarrow \emptyset, \emptyset, \emptyset,$
    **while** $\mathcal{O}^{0=\text{repeat}}_{\text{HQC}}(\mathbf{c}' + \mathbf{e}')$ **do** ▷ *Find an initial error pattern for block $B$*
        $\mathcal{I}^B_{\mathbf{e}'} \leftarrow \mathcal{I}^B_{\mathbf{e}'} \cup \{i\}$, where $i \xleftarrow{\$} \mathcal{I}^B$
        $\mathbf{e}'[i] \leftarrow 1$
    **end while**
    **for each** $i \in \mathcal{I}^B_{\mathbf{e}'}$ **do** ▷ *Minimize the error pattern*
        $\mathbf{e}'[i] \leftarrow 0$ ▷ *Unflip bit in error pattern*
        **if** $\mathcal{O}^{1=\text{repeat}}_{\text{HQC}}(\mathbf{c}' + \mathbf{e}')$ **then**
            $\mathcal{I}^B_0 \leftarrow \mathcal{I}^B_0 \cup \{i\}$ ▷ *Satisfied parity check, add $i$ to $\mathcal{I}^B_0$*
            $\mathbf{e}'[i] \leftarrow 1$ ▷ *Restore bit in error pattern*
        **end if**
    **end for**
    **for each** $i \in \left(\mathcal{I}^B \setminus \mathcal{I}^B_{\mathbf{e}'}\right)$ **do** ▷ *Find unsatisfied parity checks*
        **if** $\mathcal{O}_{\text{HQC}}(\mathbf{c}' + \mathbf{e}')$ **then**
            $\mathcal{I}^B_1 \leftarrow \mathcal{I}^B_1 \cup \{i\}$ ▷ *If found, store in $\mathcal{I}^B_1$*
        **end if**
    **end for**
    Select rows $i \in (\mathcal{I}^B_0 \cup \mathcal{I}^B_1)$ from $\mathbf{H}_{\text{ini}}$ and add to $\mathbf{H}'$
    Construct $\mathbf{H} = [\mathbf{H}'|\mathbf{I}]$
    $\mu \leftarrow \mu \mid 0^{\#\{\mathcal{I}^B_0\}} \mid 1^{\#\{\mathcal{I}^B_1\}}$
    $\mathbf{y} \leftarrow \text{Decode}_{\mathbf{H}}(\mu)[0..n]$ ▷ *Decoder returns the error vector*
    **if** $\mathbf{y}$ correct **then** ▷ *Try to decrypt a valid message*
        **return** $\mathbf{y}$
    **end if**
**end loop**

Fig. 2: HQC new attack algorithm. $\mathcal{O}^{0=\text{repeat}}_{\text{HQC}}$ denotes a PC oracle which is repeated as necessary (determined by empirical study) to achieve better than nominal error rate in the case of decoding failure; decoding successes are never repeated. $\mathcal{O}^{1=\text{repeat}}_{\text{HQC}}$ works in a similar but opposite fashion.

*An aside on oracle accuracy.* The LDPC code helps with recovery from bad oracle decisions. However, the stateful nature of the new algorithm can cause certain poor oracle decisions to propagate and result in the algorithm ending up in a bad state. Such errors occur naturally more often for less accurate oracles. We compensate for these effects by introducing extra confirmation calls to those oracle decisions which are most sensitive. These are denoted in Figure 2 by $\mathcal{O}_{\mathrm{HQC}}^{r=\mathrm{repeat}}$, where $r \in \{0, 1\}$ indicates which Oracle outputs are repeated for confirmation. $\mathcal{O}_{\mathrm{HQC}}^{0=\mathrm{repeat}}$ means decoding failures are confirmed but decoding successes are not. The number of repeated oracle calls is determined by empirical study.

After finding an error pattern resulting in decoding failure, the next step is to reduce the number of flipped bits, in block $B$. The goal is to find the minimal pattern that still results in a decoding failure. We do this by unflipping each of the flipped bits $i \in \mathcal{I}_{\mathbf{e}'}^B$ in block $B$. This results in one of two cases:

1. If we get a decoding success we record it in $\mathcal{I}_0^B$ for later use, undo the flip and then move on to select another bit $i \in \mathcal{I}_{\mathbf{e}'}^B$.
2. If we still get a decoding failure we try again with another flipped bit $i \in \mathcal{I}_{\mathbf{e}'}^B$.

Once we have run out of flipped bits in $\mathcal{I}_{\mathbf{e}'}^B$ to check, we have achieved a minimal bit pattern in $\mathcal{I}_0^B$ for block $B$ that results in decoding failure. That is, the set $\mathcal{I}_0^B$ contains those bits that result in a decoding success if any are unflipped. Conversely, when flipped, they have been unambiguously shown to increase the noise for the RMRS decoder. All bits in $\mathcal{I}_0^B$ can therefore reliably be assumed to correspond to a satisfied parity check. So, for each bit $i \in \mathcal{I}_0^B$ we construct[5] our sub matrix $\mathbf{H}'$ by the selection of row $i$ of $\mathbf{H}_{\mathrm{ini}}$.

Working from the minimal decoding failure pattern ($\mathbf{e}'[i] = 1 \,\forall\, i \in \mathcal{I}_0^B$ and $\mathbf{e}'[i] = 0 \,\forall\, i \notin \mathcal{I}_0^B$) for RM block $B$ we can now flip bits that so far have been left untouched ($i \notin \mathcal{I}_{\mathbf{e}'}^B$), one at a time. For each flip, if it results in a decoding success, then we record it in $\mathcal{I}_1^B$. Such a bit must mean that by flipping it we reduce the noise that the RMRS decoder has to handle. Therefore, this bit can be reliably assumed to correspond to an unsatisfied parity-check equation, or a '1' in the vector $\mathbf{h}_{\mathrm{ini}}\mathbf{y}$. When all bits have been tested we extend our sub matrix $\mathbf{H}'$ by the selection of all rows $i \in \mathcal{I}_1^B$ of $\mathbf{H}_{\mathrm{ini}}$.

At this time in the algorithm, $r$ number of parity-check equations have been collected in $\mathbf{H}'$. The remaining step is to construct parity-check matrix $\mathbf{H} = \left[\mathbf{H}'_{r\times n} | \mathbf{I}_{r\times r}\right]$ and a message vector

$$\mu = \left[\mathbf{0}^n | \mathbf{0}^{\#\{\mathcal{I}_0^{B^0}\}} | \mathbf{1}^{\#\{\mathcal{I}_1^{B^0}\}} \ldots | \mathbf{0}^{\#\{\mathcal{I}_0^{B^t}\}} | \mathbf{1}^{\#\{\mathcal{I}_1^{B^t}\}}\right] \tag{9}$$

in such a way that we have $n$ zeroes, each representing an unknown bit-value of $\mathbf{y}$ to be recovered. The message is appended by the following redundancies: a single 0 for each satisfied parity-check equation hitherto selected ($i \in \mathcal{I}_0^B$) and a 1 for each unsatisfied parity check ($i \in \mathcal{I}_1^B$). We do this for all $t$ blocks $B$ that have so far been selected.

[5] or extend if this is not the first selected block/iteration of the algorithm

We try to decode the message $\mu$ and recover $\mathbf{y}$ from the first $n$ bits. We use $\mathbf{H}$ as input and a suitable decoder such as sum-product or the min-sum approximation.

If the decoding is not successful we unflip all bits in block $B$ and unflip all other blocks. Then we restart the algorithm (using the same ciphertext) and select another block. The old $\mathcal{I}_0^B$ and $\mathcal{I}_1^B$ are saved and re-used in the next decoding attempt. We continue until successful.

In some cases (for less accurate oracles) one might still fail to decode even after all outer RM blocks have been exhausted. In such cases, one can simply save $\mu$ and $\mathbf{H}'$ and continue extending them by restarting the algorithm.

## 6 Experiments

In this section, we show the results of simulations and real-world experiments for Kyber and HQC.

### 6.1 Masked Kyber

**Software simulations** We introduce software simulations, where we fix the accuracy $\rho$ of each oracle $\mathcal{O}_i$ to be the same.

The attack improves as the weight of the rows in the parity matrix increases. However, the decoding time increases exponentially with it. In the course of experiments, we found that the value $w = 2$ works best, i.e., the parity-check matrix consists of negacyclic matrices with row weight 2. For Kyber-768, this means that each check variable is the sum of 6 secret coefficients.

The three main parameters of the attack are $m_0$, $m_1$ and $m_2$, where $m_0$ and $m_2$ are the lengths of the inner code for the secret coefficients and the check variables, resp., $m_1$ is the number of blocks of check variables. Recall that Kyber-768 has 3 blocks of 256 secret coefficients, and we assume that from one power trace we get information about all 256 message bits. This means that we need $3m_0$ and $m_1 \cdot m_2$ traces to get the distributions for secret coefficients and check variables, respectively. The interested reader is referred to Tables A.1 and A.2 for the actual codes used in the simulation and in the real attack.

We evaluate our methodology against the majority voting technique, a conceptually simple coding approach that can be considered as a repetition code. Majority voting is a typical approach to ensure that a single secret coefficient can be recovered with high accuracy. This approach has been selected as the baseline attack method due to its relevance as the most frequently used coding scheme for attacking Kyber in previous literature (e.g., in [41]). For majority voting, we choose the code as in Table 3 and use $t$ votes, i.e., the actual code is repeated $t$ times. We run 1000 tests and compute the average number of wrong secret coefficients, the attack is considered successful if this number is less than 1. For our approach, we choose $m_0$, $m_1$, and $m_2$ such that the total number of traces is minimized and the average number of errors is close to majority voting. We run 100 tests, and all tests are done with randomly generated secret keys. The results for a wide range of accuracy levels are shown in Table 4.

Table 4: Comparison with the majority voting for full-key recovery. $t$ is the number of votes cast, values in the brackets are $m_0$, $m_1$ and $m_2$, resp.

| $\rho = 0.995$ | Number of traces | Average number of errors |
|---|---|---|
| Majority Voting ($t = 3$) | 27 (ref) | 0.21/768 |
| Our Method $(2, 1, 4)$ | 10 ($-63\%$) | 0.37/768 |
| $\rho = 0.95$ | Number of traces | Average number of errors |
| Majority Voting ($t = 7$) | 63 (ref) | 0.47/768 |
| Our Method $(3, 4, 2)$ | 17 ($-73\%$) | 0.16/768 |
| $\rho = 0.9$ | Number of traces | Average number of errors |
| Majority Voting ($t = 11$) | 99 (ref) | 0.67/768 |
| Our Method $(4, 3, 4)$ | 24 ($-75.8\%$) | 0.46/768 |

**Real-world experiments** We conduct our experiments in the ChipWhisperer toolkit, including the ChipWhisperer-Lite board, the CW308 UFO board, and the CW308T-STM32F4 target board with a 32-bit ARM Cortex-M4 CPU. We target the `mkm4` library[6] in [22] implementing a first-order masked version of Kyber. The library is compiled using the -O3 optimization level, which is typically harder to attack [41, 29]. The target board is run at 24 MHz, and the traces are sampled at 24 MHz.

```
masked_poly_frommsg(uint16_t poly[2][256], uint8_t msg[2][32])
... /* initialization */
for i = 0 to 31 do
    for j = 0 to 7 do
        mask = -((msg[0][i] >> j) & 1)
        poly[0][8*i+j] += mask & ((KYBER_Q+1)/2)
    end for
end for
for i = 0 to 31 do
    for j = 0 to 7 do
        mask = -((msg[1][i] >> j) & 1)
        poly[1][8*i+j] += mask & ((KYBER_Q+1)/2)
    end for
end for
...
```

Fig. 3: The attacked function in KYBER.CPAPKE.Enc() (from [22])

We attacked the function `masked_poly_tomsg` in the first draft of the work and it was the first power analysis attack on an open-source masked implementation of Kyber, as far as we know. Then we switched to the function

[6] `https://github.com/masked-kyber-m4/mkm4`

Table 5: Accuracy of recovering particular bit for models. Device $D_1$ is the profiling device, and $D_2$ is the device to be attacked.

| Device | $\rho_0$ | $\rho_1$ | $\rho_2$ | $\rho_3$ | $\rho_4$ | $\rho_5$ | $\rho_6$ | $\rho_7$ |
|---|---|---|---|---|---|---|---|---|
| $D_1$ | 0.9651 | 0.9986 | 0.9985 | 0.9985 | 0.9992 | 0.9995 | 1.0000 | 1.0000 |
| $D_2$ | 0.9390 | 0.9811 | 0.9923 | 0.9023 | 0.9654 | 0.8940 | 0.9404 | 0.9873 |

Table 6: Real-world attack results on the first-order masked Kyber-768. We performed 100 runs of the attack with a random secret key for each run.

| | Number of traces | Average number of errors |
|---|---|---|
| Majority Voting ($t = 11$) | 99 | 0.34/768 |
| Our Method $(2, 2, 3)$ | 12 | 0.82/768 |

`masked_poly_frommsg` similarly to [11]. With this approach, real oracles from side-channel leakages have better accuracy, leading to a lower amount of traces.

The function `masked_poly_frommsg` (shown in Figure 3) maps each masked polynomial coefficient to a corresponding message bit during decapsulation. In one loop the function works on the message bits XORed with random bits; on the other loop it works with these random bits themselves. Obtaining a power trace for these two loops allows us to retrieve information about all message bits and implement the FD oracles $\mathcal{O}_i$.

The attack scenario is the same as in [29]. First, there is the profiling stage during which, using the profiling device $D_1$, we collect 100,000 power traces of the function `masked_poly_frommsg`. It is done by generating a random message which is encrypted using the device's public key, the resulting ciphertext is passed to the measured by the ChipWhisperer decapsulation function. Each byte of the message is computed in the same way, and the power traces corresponding to each byte are similar. Thus, we can train only 8 neural network models, one for each bit of the byte. Models are trained for up to 100 epochs. The interested reader is referred to Table A.3 for the architecture of the model.

Each of the 8 models simulates the 32 oracles $\mathcal{O}_{i+8j}$, $j = 0, \ldots, 31$, with some accuracy $\rho_i$. The oracle behaves like a binary symmetric channel with success probability $\rho_i$, but the model provides soft values, which can be treated as the probabilities of output being 1 or 0 from the model's perspective. Thus, the real-world attack is more powerful since there is more information we can work with.

After the profiling stage, there is the attacking stage. The assumption is that the attacker has access for a (relatively) short period of time to a similar device $D_2$. After collecting power traces for decapsulation on chosen ciphertexts, the attacker's goal is to recover the key using the trained models. The Table 5 shows the accuracy $\rho_i$ of recovering $i^{\text{th}}$ bit for devices $D_1$ and $D_2$.

The experimental results (shown in Table 6) with the average oracle accuracy of 0.9502 are better than the simulation results with an accuracy of 0.95. There are two reasons for this: (1) real models provide soft values, making the attack

more powerful; (2) In the simulation, the accuracy of each bit is the same, but for our LDPC approach, it is more beneficial for some bits to be more reliable than others.

On the other hand, the success of majority voting approach depends on the worst bit position. In other words, in the real world majority voting works worse since the bottleneck is the worst bit. The real attack with accuracy from Table 5 uses $t = 11$ votes, i.e. in total we need 99 traces (instead of 63 as in Table 4). In this case, our framework uses 86% fewer traces.

### 6.2 HQC

In order to test the new attack strategy against HQC it is advantageous to make as close to an apples-to-apples comparison as possible against the results of [16]. To this end, we model the PC oracle as follows; The success probability for an oracle query is determined by $\rho_0$ and $\rho_1$, which are the probabilities of correctly classifying decoding failures and decoding successes, respectively. For the case of the ideal HQC timing oracle used in [16] these values are listed in Table 7 and correspond to $\rho_0 = \rho_f$ and $\rho_1 = \rho_s$. We label the ideal oracle $\mathcal{O}^{\text{ideal}}_{\text{HQC}}$.

Table 7: Ideal HQC timing oracle, $\mathcal{O}^{\text{ideal}}_{\text{HQC}}$, as modelled with $\rho_f$ and $\rho_s$.

| Real \ Reported as | decoding failure | decoding success |
|---|---|---|
| decoding failure | $\rho_f = 0.9942$ | $1 - \rho_f = 0.0058$ |
| decoding success | $1 - \rho_s = 0$ | $\rho_s = 1.0$ |

Simulating real-world attacks with noisy measurements can be done by selecting other values of $\rho_0$ and $\rho_1$. For simplicity, we introduce $\rho$ as a single representative value of PC oracle accuracy, where $\rho = \rho_0 = \rho_1$. We label the corresponding oracle $\mathcal{O}^{\rho}_{\text{HQC}}$.

By empirical study (see Figure A.5) we have selected a row weight of $W = 50$ in the constructed LDPC code (for hqc-128). This is close to the upper limit of our code generation algorithm. Using a bigger $W$ would occasionally require a more advanced algorithm with backtracking of the random walk. Regardless, the decoding appears to suffer in reliability for values $W > 50$. Smaller values of $W$ require more parity checks and thus make the attack slower.

Some interesting $\rho$ values, corresponding to real attacks, are {1.0, 0.995, 0.95, 0.9}. In Figure 4 we show the results of simulations using the various oracle models we have described so far. The results for $\mathcal{O}^{\text{ideal}}_{\text{HQC}}$ indicate an 86.6 times improvement over the original attack [16].

We have validated our attack by running a real timing oracle on a Ubuntu 20.04 LTS laptop with Intel Core i5-7200@2.50GHz. Measurement noise was reduced by turning of hyper-threading and by running in recovery mode. We used $2^{18}$ measurements to generate a profile, first of a decoding success and

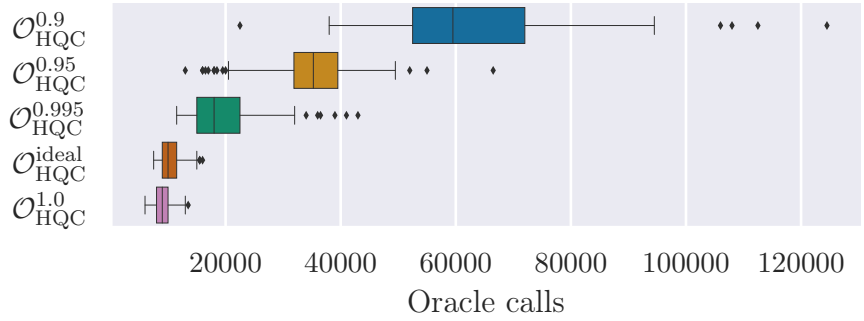


Fig. 4: Experiment for hqc-128. The median number of oracle calls for successful key recovery, are 59500, 35250, 18000, 10000, and 9000 respectively for the listed oracles. For each oracle model, 100 key-recovery simulations ran to completion.

again of a decoding failure. Measuring 8 decapsulations resulted in an oracle accuracy of $\rho_{\mathcal{O}^{\mathrm{real}}_{\mathrm{HQC}}} = 0.951$ as determined by 1000 trials. The simulated results for $\mathcal{O}^{0.95}_{\mathrm{HQC}}$ indicate a real-life key recovery attack of hqc-128 can be done by measuring $2^3 \times 35250 \approx 2^{18}$ decapsulation calls.

### 6.3 Discussions

In this section, we present discussions on the performance of the new SCA-LDPC framework, including its information-theoretical advantages and limitations. Furthermore, we compare the SCA-LDPC framework with the inner-symbol error correction method proposed in [29] and highlight the advantages of the former.

*A non-rigorous information theoretical bound.* Assuming that a single side-channel measurement provides a certain amount of information (denoted by $\mathsf{I}$ bits), and considering the fact that there are $k$ secret symbols that are independently generated from a distribution with entropy $\mathsf{E}$ bits, it is possible to calculate a lower bound for the number of measurements required. This can be accomplished by dividing $k \cdot \mathsf{E}$ by $\mathsf{I}$. Estimation of $\mathsf{I}$ can be performed by considering each recovered message bit as a Bernoulli variable, with a specified probability $\rho_i$ of being correct. It is noteworthy that the value of $\rho_i$ may vary for different secret positions. This information-theoretical estimation is approximate in nature. It is subject to limitations arising from the simplicity of the Bernoulli model. Additionally, perfect source coding and perfect channel coding are required to match the derived lower bound. Notwithstanding these limitations, the estimation suggests the possibility of improvement, though the extent of such improvement may be constrained.

The aforementioned lower bound is equivalent to the well-known Shannon source coding bound when the accuracy of the oracle is 100%, which can be used to characterize the source compression gain. The results obtained from the

FD oracle based attack on the scheme Kyber-768 exactly meet the lower bound of 7 traces. Conversely, for the PC oracle based attack on hqc-128, 1324 parity checks were required, a factor of 2.1 times the lower bound of 628 checks.

It has been observed that the difference between the simulated results and the lower bound increases as the oracle accuracy decreases. For example, in the case of the PC oracle based attack on hqc-128, when the oracle accuracy drops to 0.95, the ratio of the simulated parity checks to the lower bound increases to approximately 2.7, as calculated by 2396/880. For the FD oracle based real-world attack on Kyber-768, based on the message recovery accuracy data presented in the second row of Table 5, the lower bound was determined to be 9, which is slightly lower than the 12 traces utilized in the actual attack.

*Limitations.* Despite the remarkable reduction of necessary side-channel measurements, a gap remains between actual performance and our non-rigorous information-theoretical lower bound. This gap may be attributed to the requirement of an extremely long codeword, potentially in the range of tens of millions of bits, for the LDPC codes to approach optimality. Additionally, it may be a result of the simplicity and inadequacy of our current code-construction method. More sophisticated LDPC code construction techniques could further reduce the required number of measurements.

*Our method vs. inner-symbol error correction.* The new SCA-LDPC framework utilizes a system of sparse parity checks to interconnect all the secret symbols. As a result, accurately determined symbols can be utilized to rectify incorrectly determined symbols, categorizing this method as inter-symbol error correction. On the other hand, the method presented in [29] falls under the category of inner-symbol error correction, as the utilization of extended Hamming codes increases the possibility of recovering individual secret symbols, which all must be recovered independently.

Both methods can be applied to the FD oracle based attack on lattice-based schemes in a non-adaptive attack model. However, the inner-symbol error correction method presented in [29] offers no source compression gain and has inferior error correction capabilities. For instance, it is demonstrated in [29] that for a platform with an average message bit recovery rate of 0.972, 216 traces, or $9 \times 24$, are required to recover the long-term secret key of a masked Saber implementation. We utilize the detailed message bit recovery rates recorded in Table 20 of [29] to calculate the corresponding lower bound, which is determined to be 10 traces. This demonstrates a significant gap of 21.6 between the actual performance in a real-world scenario and the calculated lower bound. While there is no guarantee that the non-rigorous lower bound will always be attainable, the small ratio of 1.33, or 12/9, for our SCA-LDPC attack on Kyber, illustrates the superior efficiency of our method in terms of the required number of traces.

The substantial improvement of the new SCA-LDPC framework can be attributed to various factors, such as the utilization of soft information in the real-world attack that we conducted. The dominant reason is that all the secret

symbols are interconnected and correlated, and redundant symbols are introduced, allowing for the effective handling of a significant number of symbol-level errors. On the contrary, in the inner-symbol error correction method, all the symbols (e.g., 768 symbols in the Saber case) are independent and needs to be successfully recovered, thus precluding the tolerance of any symbol-level errors. Last, in a real-world attack scenario, several secret positions typically have a higher chance of containing errors, which can be effectively corrected through the inter-symbol approach, but may prove to be a bottleneck for the inner-symbol method where all symbols must be correctly identified independently.

## 7 Concluding Remarks and Future Work

From coding theory, we have presented a generic framework for key-recovery side-channel attacks on CCA-secure post-quantum encryption/KEM schemes. Our design philosophy is to employ randomly generated LDPC codes with efficient decoding to connect secret coefficients, which introduces additional benefits of source compression and error correction. We presented simulation results and real-world experiments on the main lattice-based KEM Kyber and the code-based KEM HQC. The new attack framework can significantly improve the state-of-the-art in terms of the required number of side-channel measurements. An explanation for the substantial improvements is that LDPC codes are considered to have near-optimal performance from an information-theoretic standpoint.

The sample complexity of the new attack framework can be improved further by (i) employing a more advanced code-construction method with improved decoding performance or by (ii) heavy post-processing such as lattice-reduction or information-set decoding. An intriguing area of study is to utilize sophisticated coding-theoretical methods [38], such as density evolution or EXIT charts, to carry out efficient and precise security assessments against proposed attacks.

The new attack framework can be easily applied to several other important KEM candidates in the NIST PQ project such as FrodoKEM [28] and Saber [9]. It is interesting to investigate its further applications in NTRU [7] and NTRU prime [6]. Last, the new attack framework shows the need for countermeasures such as constant-time implementations or higher-order masked implementations. Our next step is to evaluate the security of higher-order masked implementations for Kyber, as Kyber is a future NIST standard. The very recent work [11] demonstrated a high probability of success in recovering message bits from a masked Kyber implementation of up to the fifth order. In conjunction with our simulation results, this suggests higher-order masked implementations may still be highly susceptible to the present attack framework.

## Acknowledgement

This work was supported by the Swedish Research Council (grant numbers 2019-04166 and 2021-04602); the Swedish Civil Contingencies Agency (grant number

2020-11632); the Swedish Foundation for Strategic Research (Grant No. RIT17-0005); and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations and simulations were partly enabled by resources provided by LUNARC.

## References

1. Nist post-quantum cryptography standardization. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization, accessed: 2018-09-24
2. Aguilar Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Persichetti, E., Zémor, G., Bos, J.: HQC. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
3. Albrecht, M.R., Bernstein, D.J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., von Maurich, I., Misoczki, R., Niederhagen, R., Paterson, K.G., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Tjhai, C.J., Tomlinson, M., Wang, W.: Classic McEliece. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
4. Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Gueron, S., Guneysu, T., Aguilar Melchor, C., Misoczki, R., Persichetti, E., Sendrier, N., Tillich, J.P., Zémor, G., Vasseur, V., Ghosh, S.: BIKE. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
5. Backlund, L., Ngo, K., Gärtner, J., Dubrova, E.: Secret key recovery attacks on masked and shuffled implementations of crystals-kyber and saber. Cryptology ePrint Archive, Paper 2022/1692 (2022), https://eprint.iacr.org/2022/1692, https://eprint.iacr.org/2022/1692
6. Bernstein, D.J., Brumley, B.B., Chen, M.S., Chuengsatiansup, C., Lange, T., Marotzke, A., Peng, B.Y., Tuveri, N., van Vredendaal, C., Yang, B.Y.: NTRU Prime. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
7. Chen, C., Danba, O., Hoffstein, J., Hulsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P., Whyte, W., Zhang, Z., Saito, T., Yamakawa, T., Xagawa, K.: NTRU. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
8. Dachman-Soled, D., Ducas, L., Gong, H., Rossi, M.: LWE with side information: Attacks and concrete security estimation. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 329–358. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_12
9. D'Anvers, J.P., Karmakar, A., Roy, S.S., Vercauteren, F., Mera, J.M.B., Beirendonck, M.V., Basso, A.: SABER. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
10. D'Anvers, J.P., Tiepelt, M., Vercauteren, F., Verbauwhede, I.: Timing attacks on error correcting codes in post-quantum secure schemes. IACR Cryptology ePrint Archive **2019**, 292 (2019)

11. Dubrova, E., Ngo, K., Gärtner, J.: Breaking a fifth-order masked implementation of crystals-kyber by copy-paste. Cryptology ePrint Archive, Paper 2022/1713 (2022), https://eprint.iacr.org/2022/1713
12. Forney, G.D.: Concatenated codes. Technical Report 440, MIT (1965)
13. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_34
14. Gallager, R.: Low-density parity-check codes. IRE Transactions on information theory **8**(1), 21–28 (1962)
15. Goy, G., Loiseau, A., Gaborit, P.: A new key recovery side-channel attack on hqc with chosen ciphertext. In: Cheon, J.H., Johansson, T. (eds.) Post-Quantum Cryptography. pp. 353–371. Springer International Publishing, Cham (2022)
16. 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. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(3), 223–263 (2022). https://doi.org/10.46586/tches.v2022.i3.223-263, https://doi.org/10.46586/tches.v2022.i3.223-263
17. Guo, Q., Johansson, A., Johansson, T.: A key-recovery side-channel attack on classic mceliece implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(4), 800–827 (2022), https://doi.org/10.46586/tches.v2022.i4.800-827
18. Guo, Q., Johansson, T., Nilsson, A.: A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 359–386. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_13
19. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 789–815. Springer, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53887-6_29
20. Hall, C., Goldberg, I., Schneier, B.: Reaction attacks against several public-key cryptosystems. In: Varadharajan, V., Mu, Y. (eds.) ICICS 99. LNCS, vol. 1726, pp. 2–12. Springer, Heidelberg (Nov 1999)
21. Hamburg, M., Hermelink, J., Primas, R., Samardjiska, S., Schamberger, T., Streit, S., Strieder, E., van Vredendaal, C.: Chosen ciphertext k-trace attacks on masked CCA2 secure kyber. IACR TCHES **2021**(4), 88–113 (2021). https://doi.org/10.46586/tches.v2021.i4.88-113, https://tches.iacr.org/index.php/TCHES/article/view/9061
22. Heinz, D., Kannwischer, M.J., Land, G., Pöppelmann, T., Schwabe, P., Sprenkels, D.: First-order masked kyber on arm cortex-m4. Cryptology ePrint Archive, Paper 2022/058 (2022), https://eprint.iacr.org/2022/058
23. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 341–371. Springer, Heidelberg (Nov 2017). https://doi.org/10.1007/978-3-319-70500-2_12
24. Huang, S., Sim, R.Q., Chuengsatiansup, C., Guo, Q., Johansson, T.: Cache-timing attack against hqc. Cryptology ePrint Archive, Paper 2023/102 (2023), https://eprint.iacr.org/2023/102, https://eprint.iacr.org/2023/102
25. Johansson, T., Zigangirov, K.S.: A simple one-sweep algorithm for optimal APP symbol decoding of linear block codes. IEEE Trans. Inf. Theory **44**(7), 3124–3129 (1998), https://doi.org/10.1109/18.737541

26. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (Aug 1996). https://doi.org/10.1007/3-540-68697-5_9
27. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. DSN Progress Report **42-44**, 114–116 (1978)
28. Naehrig, M., Alkim, E., Bos, J., Ducas, L., Easterbrook, K., LaMacchia, B., Longa, P., Mironov, I., Nikolaenko, V., Peikert, C., Raghunathan, A., Stebila, D.: FrodoKEM. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
29. Ngo, K., Dubrova, E., Guo, Q., Johansson, T.: A side-channel attack on a masked IND-CCA secure saber KEM implementation. IACR TCHES **2021**(4), 676–707 (2021). https://doi.org/10.46586/tches.v2021.i4.676-707, https://tches.iacr.org/index.php/TCHES/article/view/9079
30. Pearl, J.: Reverend bayes on inference engines: A distributed hierarchical approach. In: AAAI (1982)
31. Qin, Y., Cheng, C., Zhang, X., Pan, Y., Hu, L., Ding, J.: A systematic approach and analysis of key mismatch attacks on lattice-based NIST candidate KEMs. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part IV. LNCS, vol. 13093, pp. 92–121. Springer, Heidelberg (Dec 2021). https://doi.org/10.1007/978-3-030-92068-5_4
32. Rajendran, G., Ravi, P., D'Anvers, J.P., Bhasin, S., Chattopadhyay, A.: Pushing the limits of generic side-channel attacks on lwe-based kems - parallel pc oracle attacks on kyber kem and beyond. Cryptology ePrint Archive, Paper 2022/931 (2022), https://eprint.iacr.org/2022/931
33. Ravi, P., Ezerman, M.F., Bhasin, S., Chattopadhyay, A., Roy, S.S.: Will you cross the threshold for me? generic side-channel assisted chosen-ciphertext attacks on ntru-based kems. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(1), 722–761 (2022). https://doi.org/10.46586/tches.v2022.i1.722-761, https://doi.org/10.46586/tches.v2022.i1.722-761
34. Ravi, P., Roy, S.S.: Side-channel analysis of lattice-based pqc candidates. https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/seminars/mar-2021-ravi-sujoy-presentation.pdf, accessed: 2022-09-29
35. Ravi, P., Roy, S.S., Chattopadhyay, A., Bhasin, S.: Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. IACR TCHES **2020**(3), 307–335 (2020). https://doi.org/10.13154/tches.v2020.i3.307-335, https://tches.iacr.org/index.php/TCHES/article/view/8592
36. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005). https://doi.org/10.1145/1060590.1060603
37. Richardson, T., Shokrollahi, M., Urbanke, R.: Design of capacity-approaching irregular low-density parity-check codes. IEEE Transactions on Information Theory **47**(2), 619–637 (2001). https://doi.org/10.1109/18.910578
38. Richardson, T., Urbanke, R.: Modern Coding Theory. Cambridge University Press, USA (2008)
39. Schamberger, T., Holzbaur, L., Renner, J., Wachter-Zeh, A., Sigl, G.: A power side-channel attack on the reed-muller reed-solomon version of the hqc cryptosystem. Cryptology ePrint Archive, Paper 2022/724 (2022), https://eprint.iacr.org/2022/724

40. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehlé, D.: CRYSTALS-KYBER. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
41. Shen, M., Cheng, C., Zhang, X., Guo, Q., Jiang, T.: Find the bad apples: An efficient method for perfect key recovery under imperfect SCA oracles - A case study of Kyber. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(1), 89–112 (2023). https://doi.org/10.46586/tches.v2023.i1.89-112, https://doi.org/10.46586/tches.v2023.i1.89-112
42. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th FOCS. pp. 124–134. IEEE Computer Society Press (Nov 1994). https://doi.org/10.1109/SFCS.1994.365700
43. Tanaka, Y., Ueno, R., Xagawa, K., Ito, A., Takahashi, J., Homma, N.: Multiple-valued plaintext-checking side-channel attacks on post-quantum kems. Cryptology ePrint Archive, Paper 2022/940 (2022), https://eprint.iacr.org/2022/940
44. 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 Trans. Cryptogr. Hardw. Embed. Syst. **2022**(1), 296–322 (2022), https://doi.org/10.46586/tches.v2022.i1.296-322
45. Xu, Z., Pemberton, O., Roy, S.S., Oswald, D., Yao, W., Zheng, Z.: Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber. IEEE Transactions on Computers **71**(9), 2163–2176 (2022). https://doi.org/10.1109/TC.2021.3122997

# Auxiliary Supporting Material

## A Supporting Figures and Tables

**Input:**
**Output:** $sk$, $pk$
$\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{d \times d}$
$\mathbf{s} \xleftarrow{\$} \mathsf{B}_{\mu_1}^d$, where $\mathbf{s} \in \mathcal{R}_q^d$
$\mathbf{e} \xleftarrow{\$} \mathsf{B}_{\mu_1}^d$, where $\mathbf{e} \in \mathcal{R}_q^d$
$sk \stackrel{\text{def}}{=} \mathbf{s}$
$pk \stackrel{\text{def}}{=} \mathbf{A}\mathbf{s} + \mathbf{e}$

(a) KYBER.CPAPKE.KeyGen()

**Input:** $sk$, $c = (\mathbf{c}_1, c_2)$
**Output:** $\mathbf{m}$
$\mathbf{u} = \mathtt{Decomp}_q(\mathbf{c}_1, d_u)$
$v = \mathtt{Decomp}_q(c_2, d_v)$
$m = \mathtt{Comp}_q(v - \mathbf{s}^{\mathrm{T}}\mathbf{u}, 1)$

(b) KYBER.CPAPKE.Dec()

**Input:** $pk$, $m$, $r$
**Output:** $c = (\mathbf{c}_1, c_2)$
Generate $\mathbf{A} \in \mathcal{R}_q^{d \times d}$ from $pk$
$\mathbf{p} = pk$
$\mathbf{r} \xleftarrow{\$} \mathsf{B}_{\mu_1}^d$, where $\mathbf{r} \in \mathcal{R}_q^d$
$\mathbf{e}_1 \xleftarrow{\$} \mathsf{B}_{\mu_2}^d$, where $\mathbf{e}_1 \in \mathcal{R}_q^d$
$e_2 \xleftarrow{\$} \mathsf{B}_{\mu_2}$, where $e_2 \in \mathcal{R}_q$
$\mathbf{u} = \mathbf{A}^{\mathrm{T}}\mathbf{r} + \mathbf{e}_1$
$v = \mathbf{p}^{\mathrm{T}}\mathbf{r} + e_2 + \mathtt{Decomp}_q(m, 1)$
$\mathbf{c}_1 \stackrel{\text{def}}{=} \mathtt{Comp}_q(\mathbf{u}, d_u)$
$c_2 \stackrel{\text{def}}{=} \mathtt{Comp}_q(v, d_v)$
$c \stackrel{\text{def}}{=} (\mathbf{c}_1, c_2)$

(c) KYBER.CPAPKE.Enc()

Fig. A.1: KYBER.CPAPKE (simplified version)

**Input:**
**Output:** $sk$, $pk$

Generate a pseudo-random coin $z$
$(pk, sk') \stackrel{\text{def}}{=}$ KYBER.CPAPKE.KeyGen()
$sk \stackrel{\text{def}}{=} (sk', pk, \mathcal{H}(pk), z)$

(a) KYBER.CCAKEM.KeyGen()

**Input:** $pk$
**Output:** $c, K$

$m \xleftarrow{\$} \{0,1\}^{256}$
$m = \mathcal{H}(m)$
$(\bar{K}, r) = \mathcal{G}(m, \mathcal{H}(pk))$
$c =$ KYBER.CPAPKE.Enc$(pk, m, r)$
$K = \mathsf{KDF}(\bar{K}, \mathcal{H}(c))$

(b) KYBER.CCAKEM.Encaps()

**Input:** $sk$, $c$
**Output:** $K$

$m' =$ KYBER.CPAPKE.Dec$(sk, c)$
$(\bar{K}', r') = \mathcal{G}(m', \mathcal{H}(pk))$
$c' =$ KYBER.CPAPKE.Enc$(pk, m', r')$
**if** $c = c'$ **then**
  $K = \mathsf{KDF}(\bar{K}', \mathcal{H}(c))$
**else**
  $K = \mathsf{KDF}(z, \mathcal{H}(c))$
**end if**

(c) KYBER.CCAKEM.Decaps()

Fig. A.2: KYBER.CCAKEM

- Setup$(1^\lambda)$: generates the global parameters param $= (n, k, \delta, \omega, \omega_{\mathbf{r}}, \omega_{\mathbf{e}})$.
- KeyGen(param): sample $\mathbf{h} \xleftarrow{\$} \mathcal{R}_2$, the generator matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ of $\mathcal{C}$, sk $= (\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathcal{R}_2^2$ such that $\omega(\mathbf{x}) = \omega(\mathbf{y}) = \omega$, sets pk $= (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$, and returns (pk, sk).
- Encrypt(pk, $\mathbf{m}$): generates $\mathbf{e} \xleftarrow{\$} \mathcal{R}_2$, $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2) \xleftarrow{\$} \mathcal{R}_2^2$ such that $\omega(\mathbf{e}) = \omega_{\mathbf{e}}$ and $\omega(\mathbf{r}_1) = \omega(\mathbf{r}_2) = \omega_{\mathbf{r}}$, sets $\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$ and $\mathbf{v} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}$, returns $\mathbf{c} = (\mathbf{u}, \mathbf{v})$.
- Decrypt(sk, $\mathbf{c}$): returns $\mathcal{C}$.Decode$(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})$.

Fig. A.3: Description of the proposal HQC.PKE [2].

– $\mathsf{Setup}(1^\lambda)$: generates the global parameters $\mathsf{param} = (n, k, \delta, \omega, \omega_{\mathbf{r}}, \omega_{\mathbf{e}})$.

– $\mathsf{KeyGen}(\mathsf{param})$: exactly as in HQC.PKE.

– $\mathsf{Encapsulate}(\mathsf{pk})$: generate $\mathbf{m} \xleftarrow{\$} \mathbb{F}_2^k$, which will serve as the seed to derive the shared key. Derive the randomness $\theta \xleftarrow{\$} \mathcal{G}(\mathbf{m})$. Generate the ciphertext $\mathbf{c} \leftarrow (\mathbf{u}, \mathbf{v}) = \mathcal{E}.\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}, \theta)$, and derive the symmetric key $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$. Let $\mathbf{d} \leftarrow \mathcal{H}(\mathbf{m})$, and send $(\mathbf{c}, \mathbf{d})$.

– $\mathsf{Decapsulate}(\mathsf{sk}, \mathbf{c}, \mathbf{d})$: decrypt $\mathbf{m}' \leftarrow \mathcal{E}.\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$, compute $\theta' \leftarrow \mathcal{G}(\mathbf{m}')$, and (re-)encrypt $\mathbf{m}'$ to get $\mathbf{c}' \leftarrow \mathcal{E}.\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m}', \theta')$. If $\mathbf{c} \neq \mathbf{c}'$ or $\mathbf{d} \neq \mathcal{H}(\mathbf{m}')$ then abort. Otherwise, derive the shared key $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$.

Fig. A.4: Description of the proposal HQC.KEM [2].

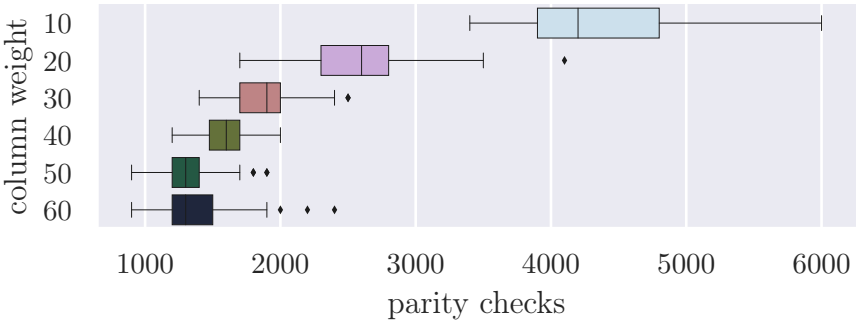


Fig. A.5: Experimental results of varying the column weight of the generated LDPC code. Weight of 50 appears to require the fewest amount of parity checks for successful key recovery.

Table A.1: The best inner codes for secret coefficients for given accuracy level $\rho$.

| $\rho$ | $m_0$ | $(k'_u, k'_v)$ | Secret coefficient | | | | |
|---|---|---|---|---|---|---|---|
| | | | -2 | -1 | 0 | 1 | 2 |
| $0.995, 0.95$ | 2 | $(630, 0)$ | 0 | 1 | 0 | 1 | 0 |
| | | $(706, 6)$ | 0 | 0 | 1 | 1 | 0 |
| $1, 0.95$ | 3 | $(630, 14)$ | 0 | 1 | 0 | 1 | 1 |
| | | $(706, 6)$ | 0 | 0 | 1 | 1 | 0 |
| | | $(706, 10)$ | 0 | 1 | 1 | 0 | 0 |
| 0.9 | 4 | $(630, 14)$ | 0 | 1 | 0 | 1 | 1 |
| | | $(706, 6)$ | 0 | 0 | 1 | 1 | 0 |
| | | $(706, 10)$ | 0 | 1 | 1 | 0 | 0 |
| | | $(630, 10)$ | 0 | 0 | 1 | 0 | 1 |

Table A.2: "Reasonable" inner codes for check variables. We assume that each check variable is the sum of 6 secret coefficients.

| $m_2$ | $(k'_u, k'_v)$ | Coefficients for check variable, $[-12, \ldots, 12]$ |
|---|---|---|
| 2 | $(180, 0)$ | 000110001110001110001100 0 |
| | $(423, 14)$ | 010100101011010100101011 0 |
| 3 | $(401, 1)$ | 010100101001011010110100 1 |
| | $(630, 11)$ | 010010100101101001010010 1 |
| | $(483, 7)$ | 010101101010101010101010 0 |
| 4 | $(636, 5)$ | 001011010010110100101101 0 |
| | $(486, 1)$ | 010101010101010101011010 1 |
| | $(139, 2)$ | 011100001111000111100001 1 |
| | $(630, 9)$ | 011010110100101101011010 0 |

Table A.3: The architecture of the neural network used for the real-world attack on Kyber.

| Layer type | (Input, output) shape | # Parameters |
|---|---|---|
| Batch Normalization 1 | (64, 64) | 256 |
| Dense 1 | (64, 64) | 4160 |
| Batch Normalization 2 | (64, 64) | 256 |
| ReLU | (64, 64) | 0 |
| Dense 2 | (64, 32) | 2080 |
| Batch Normalization 3 | (32, 32) | 128 |
| ReLU | (32, 32) | 0 |
| Dense 3 | (32, 16) | 528 |
| Batch Normalization 3 | (16, 16) | 64 |
| ReLU | (16, 16) | 0 |
| Dense 4 | (16, 1) | 17 |
| Sigmoid | (1, 1) | 0 |

Table A.4: HQC number of required parity checks for successful decoding

| $\rho$ | weight | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|---|
| $\mathcal{O}_{\mathrm{HQC}}^{0.9}$ | 50 | 103 | 3222.33 | 873.57 | 1100 | 2700 | 3000 | 3600 | 6600 |
| $\mathcal{O}_{\mathrm{HQC}}^{0.95}$ | 50 | 100 | 2396 | 629.40 | 900 | 2175 | 2400 | 2700 | 4900 |
| $\mathcal{O}_{\mathrm{HQC}}^{0.995}$ | 50 | 100 | 1623 | 546.40 | 1000 | 1200 | 1400 | 1800 | 3500 |
| $\mathcal{O}_{\mathrm{HQC}}^{\mathrm{ideal}}$ | 50 | 110 | 1365.45 | 261.41 | 900 | 1200 | 1300 | 1500 | 2100 |
| $\mathcal{O}_{\mathrm{HQC}}^{1.0}$ | 50 | 100 | 1324 | 208.47 | 900 | 1200 | 1300 | 1400 | 1900 |

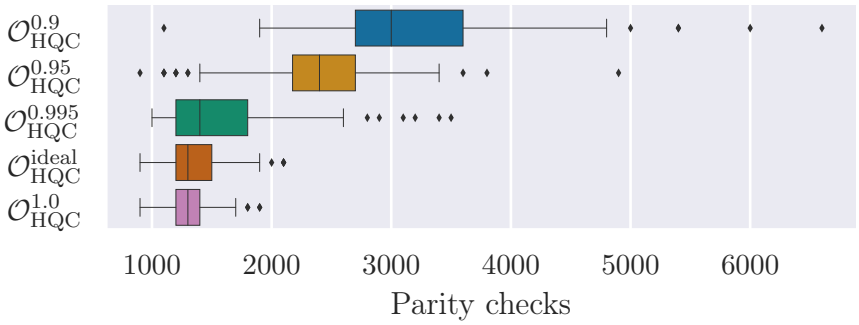


Fig. A.6: Boxplot of the previous table