Divide and Surrender: Exploiting Variable Division Instruction Timing in HQC Key Recovery Attacks

Robin Leander Schröder
Fraunhofer SIT, Darmstadt, Germany
Fraunhofer Austria, Vienna, Austria

Stefan Gast
Graz University of Technology, Austria

Qian Guo
Lund University, Sweden

Abstract

We uncover a critical side-channel vulnerability in the Hamming Quasi-Cyclic (HQC) round 4 optimized implementation arising due to the use of the modulo operator. In some cases, compilers optimize uses of the modulo operator with compile-time known divisors into constant-time Barrett reductions. However, this optimization is not guaranteed: for example, when a modulo operation is used in a loop the compiler may emit division (`div`) instructions which have variable execution time depending on the numerator. When the numerator depends on secret data, this may yield a timing side-channel. We name vulnerabilities of this kind Divide and Surrender (DaS) vulnerabilities.

For processors supporting Simultaneous Multithreading (SMT) we propose a new approach called DIV-SMT which enables precisely measuring small division timing variations using scheduler and/or execution unit contention. We show that using only 100 such side-channel traces we can build a Plaintext-Checking (PC) oracle with above 90% accuracy. Our approach might also prove applicable to other instances of the DaS vulnerability, such as KyberSlash. We stress that exploitation with DIV-SMT requires co-location of the attacker on the same physical core as the victim.

We then apply our methodology to HQC and present a novel way to recover HQC secret keys faster, achieving an 8-fold decrease in the number of idealized oracle queries when compared to previous approaches. Our new PC oracle attack uses our newly developed Zero Tester method to quickly determine whether an entire block of bits contains only zero-bits. The Zero Tester method enables the DIV-SMT powered attack on HQC-128 to complete in under 2 minutes on our targeted AMD Zen2 machine.

1 Introduction

The rise of quantum computing presents a grave threat to current cryptographic infrastructures, primarily based on problems like factoring and discrete logarithms. Shor's algorithm [55], in particular, significantly compromises these security foundations. In response, the National Institute of Standards and Technology (NIST) initiated a process in 2016 to identify and standardize quantum-resistant public-key cryptographic algorithms, focusing on developing new standards for Key Encapsulation Mechanism (KEM) and digital signatures. This initiative drew a robust response from the cryptographic community and after thorough review and feedback, NIST endorsed CRYSTALS-Kyber [53] as the primary KEM algorithm and CRYSTALS-Dilithium [44], FALCON [48], and SPHINCS+ [37] as key digital signature algorithms. This initiative marks a significant shift towards securing cryptographic practices against emerging quantum computing threats.

One notable aspect of NIST's selection is its emphasis on lattice-based algorithms like CRYSTALS-Kyber, CRYSTALS-Dilithium, and FALCON, reflecting a preference for lattice-based cryptography. However, to ensure a robust cryptographic future, NIST also prioritizes diversity in cryptographic solutions, hence its ongoing round 4 selection process included non-lattice-based KEM algorithms. This phase spotlighted code-based KEMs such as Classic McEliece [7], HQC [4], and BIKE [13], alongside a now broken isogeny-based KEM, SIKE [39]. Among these, the HQC proposal [4,5,14], based on hard decoding problems in coding theory, emerges as a promising candidate [6].

Given HQC's potential for NIST standardization, an immediate and critical focus must be directed towards scrutinizing HQC's security claims. Specifically, secure implementation strategies for HQC on various real-world platforms warrant exhaustive study.

Numerous studies, such as [15, 27–30, 36, 46, 52, 60, 62], have extensively explored HQC's security and implementation aspects, adopting a dynamic attack-and-fix methodology throughout NIST's initial evaluation rounds. The fourth-round HQC submission includes a single implementation claimed to be constant-time, suggesting resilience against timing attacks [40]—a type of attack that leverages variations in execution time to compromise security. As constant-time imple-

mentation is a critical aspect in cryptographic libraries, it is a fascinating research challenge to either validate this security claim or identify a more efficient attack strategy for potent key-recovery attacks.

In our paper, we identify a novel timing side-channel vulnerability in the implementation for HQC, originating from division operations that are not constant-time. This vulnerability enables an efficient key recovery timing attack against the claimed constant-time implementation in the designers' round 4 submission to NIST. This makes HQC one of the few known cases where division instructions are used in a cryptosystem with secret inputs and cause an exploitable timing difference. We call these vulnerabilities instances of the DaS vulnerability. Other instances are a potential fix for the Lucky13 attack [41] and KyberSlash [17, 26]. In KyberSlash similar code structure causes the compiler to emit `div` instructions where constant-time code could have been generated.

Our approach builds on the established PC oracle-based side-channel attacks, creating a PC oracle that verifies if a particular ciphertext c decrypts to a given message m. This type of attack can be traced back to the reaction attack, initially introduced by Hall et al. in 1999 [33]. Furthermore, this attack methodology has found broad applicability in the realm of post-quantum cryptography, as evidenced by works such as [19, 31, 45, 49, 50, 61], and is applicable to various side-channels such as timing, cache timing, power, and electromagnetic radiation, provided the side-channel leakage is sufficient to construct a PC oracle.

We develop the desired PC oracle using a novel side-channel, called DIV-SMT, which originates from integer modulo operations in some SMT processors such as AMD Zen+/Zen2 processors.

On the x86-64 instruction set architecture, implemented by the AMD Zen+/Zen2 processors, the `div` instruction is used to calculate both, the quotient and the modulo of the two operands. Hence, divisions and modulo operations in the source code are usually both compiled to `div` instructions. Compiler optimizations eliminate the `div` instructions in some cases. Specifically, with a divisor known at compile time, the GCC compiler might apply Barrett Reduction [16] as an optimization. However, as we show in Section 4, this optimization is sometimes not performed, even if it would be possible.

In these widely deployed high-end processors, varying numerator and divisor sizes in integer divisions can result in single cycle execution time differences. Exploiting such small timing differences is non-trivial and the `div` instructions only make up a small part of a complete decapsulation in the HQC cryptosystem. Further, advanced CPU features like out-of-order execution can render timing measurement more challenging. Thus, the SMT context becomes crucial here, allowing the attacker and victim threads to run on the same physical core with shared caches, execution units, and schedulers. This setup can create contention on execution units [8] or schedulers [25], enabling an adversary to detect when a co-located program executes a specific instruction, such as a multiplication or a division.

In contrast to these prior works, which attacked non-constant-time ECDSA and RSA implementations by detecting the points in time when specific instructions are executed through a timing side channel, our novel approach infers information about the numerators for the same division instruction. Our attack highlights a serious security risk in the HQC implementation submitted to NIST that claims to be constant-time. The claimed constant-time property is crucial for the safe integration of HQC into real-world libraries.

While avoiding non-constant-time instructions is a standard practice in constant-time implementations, division operations in modern high-end CPUs have previously seen little cryptographic research attention due to the lack of documented vulnerabilities and exploits resulting from their timing behavior. To our knowledge, our research presents the first key-recovery attack that clearly demonstrates the potential of exploiting this vulnerability on current, SMT enabled CPUs. Thus, our methodology of distinguishing different numerators in an SMT environment can offer valuable insights for security assessments beyond attacking HQC.

In December 2023, a vulnerability related to the use of modulo operations, potentially compiling into `div` instructions, was independently identified in the CRYSTALS-Kyber reference implementation by Bernstein [17] and by Tamvada, Kiefer, and Bhargavan [26]. The CRYSTALS-Kyber design team has acknowledged and purportedly rectified this issue. These groups did not present an attack strategy for exploiting this vulnerability in high-end modern CPUs, but their discoveries highlight the broad relevance and impact of our innovative attack methodology in the SMT context, as we deem it likely that our DIV-SMT methodology can also be used to exploit KyberSlash vulnerable implementations of Kyber.

Utilizing the PC oracle derived from the DIV-SMT side-channel, we further advance our research by developing a methodology that significantly reduces the number of PC oracle queries needed to extract HQC's secret key. This new approach holds significance beyond the newly discovered timing attack exploiting the DaS vulnerability for two main reasons. First, it relies on a PC oracle that can be built from various side-channel leakages. This versatility makes it suitable for a broad spectrum of PC oracle-based side-channel attacks on HQC, thus widening its applicability beyond just timing attacks. Second, since interacting with the oracle is often the primary bottleneck in such attacks, optimizing oracle calls is crucial for efficiency in most of real-world attack scenarios.

We introduce a new technique called Zero Tester for efficiently identifying through PC oracle calls whether a consecutive block, referred to as an 'inner block' in HQC's coding scheme, consists solely of zero-bits. Through the integration

Work	Oracle Calls
GHJLNS22 [28]	866 143
SHRWS22 [52]	>50 000
HSCGJ23 [36]	>50 000
SCA-LDPC [32]	10 000
Our Work	1 142

Table 1: Comparison of our attack to previous works. The reported number of idealized oracle calls required for HQC-128 key-recovery is shown.

of this method with a shifting strategy, we have the capacity to pinpoint numerous zero-bit's positions inside the secret vector using few queries. This method significantly increases the speed of full key recovery: Given that the public key already provides n linear equations for the $2n$ unknowns in HQC's secret key, identifying approximately n out of $2n$ unknown zero positions is sufficient.

Summary of Contributions. Our core contributions are:

- We unveil a novel timing side-channel vulnerability in the supposedly constant-time optimized reference implementation of the HQC round 4 submission to NIST, which is an instance of the vulnerability class named Divide and Surrender (DaS).
- We propose a new practical exploitation strategy of division timing variance through the use of DIV-SMT side-channels in SMT processors. By leveraging contention between SMT sibling threads, we detect the execution of specific instructions (`div`) and infer information about the operands processed (specifically, the count of leading zeros in the numerands). This insight lays the groundwork for creating a Plaintext-Checking (PC) oracle, enabling us to execute a key-recovery timing attack.
- We propose a novel key-recovery method for HQC using the PC oracle, significantly reducing the required number of side-channel traces. A detailed comparison with prior studies for HQC-128 key-recovery is presented in Table 1. We conduct extensive simulations to investigate the efficacy of our proposed key-recovery method, employing a simulated PC oracle across different levels of oracle accuracy. Our simulations demonstrate a significant improvement requiring only 11% to 13% of the oracle calls relative to the SCA-LDPC framework presented at ASIACRYPT 2023 [32]. Notably, this gain persists even as the oracle's accuracy decreases to 0.9.
- We demonstrate the practical applicability of our attack through evaluation on an AMD Zen2 CPU. Out of 1000 attacks conducted, three-quarters lead to successful full key recovery, achieving a median attack time of 111 seconds.

The artifact is available at https://github.com/hqc-attack/divide-and-surrender.

Responsible Disclosure. We disclosed the issue to the HQC design team in February 2023. After the release of our first draft, the scheme designers released a patched optimized reference implementation on 2024-02-23 [23]. The designers chose to manually implement the Barrett reductions (cf. Section 6.1) [35]. Furthermore, we disclosed the issue to the PQClean team in March 2023 [9]. We believe our disclosure allowed implementers to implement a presumably constant-time version of the vector sampling algorithm using explicit Barrett reductions.

Outline. The rest of this paper unfolds as follows: Section 2 provides the essential background, covering both the HQC KEM proposal and relevant aspects of CPU architectures. We present our main attack strategies in Section 3. This is followed by the introduction of a novel PC oracle in Section 4, specifically designed to exploit a division timing side-channel to challenge the constant-time claims of an implementation submitted to NIST. Experimental results are reported in Section 5, followed by a comprehensive discussion in Section 6. Section 7 draws conclusions.

2 Preliminaries

In this section, we present some preliminary background on the HQC proposal, CPU Pipelines, and SMT.

Notations. Consider $\mathbb{F}_2$ as the binary finite field, where the operations $+$ and $-$ are equivalent. The Hamming weight of a binary vector is defined as the number of non-zero entries in the vector. In the HQC scheme, we employ a cyclic polynomial ring $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$, where n is an integer in $\mathbb{Z}$. Elements in $\mathcal{R}$ can be alternatively viewed as row vectors in a vector space over $\mathbb{F}_2$. For uniform sampling from a set $\mathcal{S}$, we utilize the notation $\leftarrow\!\$\,\mathcal{S}$. Specifically, $\leftarrow\!\$\,\mathcal{R}_\omega$ indicates uniform sampling from $\mathcal{R}$ of an element with a Hamming weight of ω.

2.1 HQC

HQC [4], a code-based post-quantum IND-CCA (Indistinguishability under Chosen Ciphertext Attack) secure KEM, leverages the complexity of decoding random quasi-cyclic codes in the Hamming metric for its security. It was proposed as a KEM candidate in the fourth round of the NIST Post-Quantum Cryptography (PQC) standardization [18]. At the conclusion of the fourth round, NIST intended to standardize a single code-based KEM primitive, either HQC or BIKE. Similar to other NIST PQC Public Key Encryption (PKE)/KEM candidates, HQC proposal commences with an IND-CPA (Indistinguishability under Chosen Plaintext Attack) version, termed HQC.CPAPKE, and subsequently introduces an INC-CCA KEM, called HQC.CCAKEM, through a CCA transformation (HQC utilizes the Hofheinz-Hövelmanns-Kiltz (HHK) transformation [34]).

2.1.1 The PKE Version of HQC

The HQC.PKE algorithm comprises three sub-procedures: PKE.KeyGen, PKE.Encrypt, and PKE.Decrypt. Within PKE.KeyGen, the algorithm uniformly samples three polynomial elements—h, x, and y, with x and y maintaining fixed Hamming weights of w. The secret key is designated as (x,y) and the public key as $(h, s = x + h \cdot y)$. PKE.Encrypt begins by initializing the pseudo-random number generator (PRNG) with a seed θ making the sampling process deterministic. Subsequently, the algorithm uniformly samples the polynomials r_1 and r_2 from $\mathcal{R}$, each possessing a Hamming weight of w_r, and e with Hamming weight w_e. The ciphertext c is computed as $c = (u,v)$, where $u = r_1 + h \cdot r_2$ and $v = m\mathbf{G} + s \cdot r_2 + e$. The matrix $\mathbf{G}$ is the generator matrix of the linear code $\mathcal{C}$, which we will describe later. PKE.Decrypt employs a decoder $\mathcal{C}$.Decode$(v - u \cdot y)$, where

$$v - u \cdot y = m\mathbf{G} + s \cdot r_2 + e - (r_1 + h \cdot r_2) \cdot y = m\mathbf{G} + e' \quad (1)$$

$$e' = x \cdot r_2 - r_1 \cdot y + e \quad (2)$$

given that $s = x + h \cdot y$. If the Hamming weight of the error term e' is small (i.e., within the decoding capability of the employed decoder), the decoder could correct such an error, leading to successful decryption.

Beginning with the October 2020 release, HQC transitioned to a design incorporating a decoding strategy that utilizes a concatenated code combining an internal duplicated Reed-Muller (RM) code and an outer Reed-Solomon (RS) code. The resultant code produces a publicly known generator matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n_1 n_2}$, where $k = 8k_1$.

The specifics of the HQC parameters are detailed in Table 2. HQC computations occur within the ambient space $\mathbb{F}_2^n$, with any remaining $n - n_1 n_2$ positions of no value discarded. The concatenated code, $\mathcal{C}$, merges an internal duplicated RM code with an outer RS code. The internal duplicated RM code possesses parameters $[n_2, 8, n_2/2]$, and the outer RS code is defined as $[n_1, k_1, n_1 - k_1 + 1]$.

During the encoding process, a message $m \in \mathbb{F}_{2^8}^{k_1}$ translates into $m_1 \in \mathbb{F}_{2^8}^{n_1}$ by the outer Reed-Solomon code. The internal duplicated Reed-Muller code then encodes each byte $m_{1,i}$ into $\bar{m}_{1,i} \in \mathbb{F}_2^{n_2}$, where $0 \leq i < n_1$. Consequently, we attain $m\mathbf{G} = (\bar{m}_{1,0}, \cdots, \bar{m}_{1,n_1-1})$ with dimension $n_1 n_2$.

To decode $V = v - u \cdot y$, $V \in \mathbb{F}_2^{n_1 n_2}$ is partitioned into n_1 blocks, denoted as $V = (V_0, \cdots, V_{n_1-1})$, with each $V_i \in \mathbb{F}_2^{n_2}$ defined as an 'inner block', where $0 \leq i < n_1$. Each V_i undergoes decoding by the internal duplicated Reed-Muller code into $\bar{V}_i \in \mathbb{F}_2^8$, where $0 \leq i < n_1$. Thereafter, $\bar{V}$ is compiled as a string of $8n_1$ bits, denoted as $(\bar{V}_0, \cdots, \bar{V}_{n_1-1})$. For each $i \in [0, n_1)$, $\bar{V}_i$ is termed an 'internal codeword'. It is observable that $\bar{V}$ is a noisy codeword of the outer Reed-Solomon code, which can be decoded into k_1 elements over $\mathbb{F}_{256}$ and transformed into k_1 message bytes.

Input: pk
Output: K, $c = (u,v)$, d

1: $m \leftarrow\$ (\mathbb{F}_2^k)$
2: salt $\leftarrow\$ \mathbb{F}_2^{128}$
3: $\theta \leftarrow \mathcal{G}(m \| \mathsf{pk} \| \text{salt})$
4: $c \leftarrow$PKE.Encrypt(pk, m, θ)
5: $K \leftarrow \mathcal{K}(m, c)$

(a) KEM.Encaps

Input: sk $= (x,y)$, $c = (u,v)$, salt
Output: K

1: $m' \leftarrow$PKE.Decrypt(sk, c)
2: $\theta' \leftarrow \mathcal{G}(m' \| pk \| \text{salt})$
3: $c' \leftarrow$PKE.Encrypt$(\mathsf{pk}, m', \theta')$
4: **if** $m' = \perp \ \vee \ c \neq c'$ **then**
5: $\quad K \leftarrow \mathcal{K}(\sigma, c)$
6: **else**
7: $\quad K \leftarrow \mathcal{K}(m', c)$

(b) KEM.Decaps

Figure 1: HQC.CCAKEM

2.1.2 The KEM Version of HQC

The HQC KEM, depicted in Figure 1, is constructed from the HQC PKE scheme using an HHK transform. Crucially the encryption is now de-randomized and the decapsulation performs a re-encryption step to verify the validity of the ciphertext. Additionally the decapsulation invokes two distinct cryptographic hash functions: $\mathcal{G}$ and $\mathcal{K}$. The seed θ is derived from the message m, the public key and a random 128-bit salt. The seed θ and the inputs that are used to derive it are vital to the exploitation of the scheme, as it determines the side-channel behavior of a ciphertext (cf. Section 4).

2.2 CPU Pipelines

Modern superscalar CPUs execute multiple instructions in parallel [22]. The CPU frontend decodes instructions into micro-ops (µops) and forwards them to the backend via a dedicated dispatch buffer. The CPU backend has multiple execution units, processing µops in parallel. Simple µops, (e.g., additions of two registers) can typically be executed by multiple execution units, whereas more complex µops (e.g., divisions) require a specialized execution unit [12, 38]. The execution units are controlled by one [38] or multiple [10–12] schedulers that retrieve µops from the dispatch buffer and determine which µops are ready for execution, based on their operand dependencies, enabling out-of-order execution. The results of the executed µops become architecturally visible after they retire in the instruction stream order.

	RS-S			Duplicated RM						
Instance	n_1	k_1	d_{RS}	Mult.	n_2	d_{RM}	$n_1 n_2$	n	ω	$\omega_r = \omega_e$
HQC-128	46	16	31	3	384	192	17664	17669	66	75
HQC-192	56	24	33	5	640	320	35840	35851	100	114
HQC-256	90	32	49	5	640	320	57600	57637	131	149

Table 2: The HQC parameter sets [4]. The base Reed-Muller code is the first-order $[128, 8, 64]$ Reed-Muller code.

2.3 Simultaneous Multithreading (SMT)

CPUs reach their maximum performance if they utilize all execution units simultaneously. Usually, this is not achieved with a single instruction stream. Therefore, many modern CPUs execute multiple instruction streams on the same core, sharing caches, execution units and schedulers. Most Intel and AMD CPUs support two SMT threads on the same core [12, 38]. Some Power10 CPUs can support up to 8 SMT threads per core [2]. While this increases performance, it also enables multiple side channels [57, 58]. Particularly, contention on execution units [8] or schedulers [25] enables an adversary to observe that a co-located program executes a specific machine instruction. This in turn allows an adversary to recover secret keys when attacking non-constant-time ECDSA and RSA implementations [8, 25].

3 New Key-Recovery Attack

In this section, we present a novel key-recovery attack on HQC using a PC oracle. We begin with a general description of the threat model and provide a comprehensive overview of the key recovery process used in such attacks. Following this, we review several existing attacks in detail in this context. Finally, we describe our new method, which efficiently identifies blocks in the secret vector that have a zero Hamming weight.

3.1 The Generic Threat Model

In our study, we investigate a side-channel-assisted chosen-ciphertext attack model targeting HQC's decapsulation algorithm. Here, the attacker crafts ciphertexts and observes side-channel information emanating from the targeted device. This could include timing, cache-timing, or even power and electromagnetic leakages. Our general threat model posits that the attacker can construct a PC oracle $\mathcal{O}^{\rho}_{\text{HQC}}$ from side-channel leakage. This oracle confirms or denies whether $\mathsf{PKE.Decrypt}(sk, c) \stackrel{?}{=} m$, where c represents the ciphertext and m a message vector. This oracle has an accuracy level of ρ, meaning it returns a correct decision with a probability of ρ and an incorrect one with a probability of $1-\rho$. Leveraging such an oracle, our focus shifts to methods for full secret key recovery. We assume that the adversary can choose both the ciphertext and the message.

The above framework outlines our generic threat model. For any specific attack scenario, a more detailed threat model must be articulated, as the adversary's methodology for constructing the PC oracle could vary substantially depending on the concrete attack. For instance, in Section 4.2.1, we elaborate on the threat model pertinent to our newly identified timing attacks originating from division timing side-channels, where both the adversary and the victim need to run in the SMT setting.

3.2 High-Level Overview of Key Recovery

We present an end-to-end description of PC oracle-based attacks on HQC. These attack strategies exploit decoding failures, which leak key information and are detected through a PC oracle built from side-channel leakages. As described in Section 2.1.1, the ciphertext (u, v) is computed using the message m, intermediate random vectors r_1, r_2, e, and public parameters h, s, and $\mathbf{G}$. The attacker strategically assigns values to m, r_1, r_2, and e to engineer ciphertexts (u, v) that exhibit a specific property. These chosen ciphertexts are then submitted to the device for decapsulation, during which the attacker monitors the side-channel leakages to construct the PC oracle.

Since decoding failures leak information about the error contained within a ciphertext, the attacker's task then becomes crafting ciphertexts whose error yields information about the secret key. In particular, since the input to the decoder $\mathcal{C}.\text{Decode}(\cdot)$ is $v - u \cdot y$, the decoding result is linked to the secret vector y, and also to x. via the public h and the relation $s = x + h \cdot y$. Upon determining the decoding outcome—success or failure—via the PC oracle, the attacker can recover the key through methods such as bit-by-bit determination [28], enumerating patterns within an inner RM block [36], or querying sparse parity checks of several bits [32]. These strategies will be elaborated on in Section 3.3.

All of these attack strategies operate under a common assumption. They all presume that the decryption outcome hinges on a single inner block decoding of the RM code. In other words, the attacks have already introduced enough errors to push the outer RS decoding to its boundary for correct decoding. Thus, corrupting a single additional RM block

causes a decoding error of the entire concatenated code construction. This process is an optimization that is not strictly necessary for exploitation, but significantly reduces the number of online queries.

3.3 Relevant Attacks

We briefly review three relevant attacks, each discussed in the following references: [28], [36], and [32].

3.3.1 The Key-Recovery Attack by Guo et al.

Guo et al. [28] propose the following method: the attacker sets r_1, r_2 and e in such a way that the error that the decoder must correct is simply the secret y. This is done by setting r_1 to 1 (the multiplicative identity of R) and both r_2 and e to 0 (the zero vector). The foundational ciphertext is then

$$C_0 = (u, v) = (1, m\mathbf{G}). \tag{3}$$

Passing this ciphertext into PKE.Decrypt reveals that it will run the decoder on $v - u \cdot y$, which simplifies to $m\mathbf{G} - y$. Thus, the decoder receives the encoded message with the error y, which is part of the secret key.

As in all of the following attacks, the attacker then corrupts RM blocks until one more corrupted RM block will lead to a decoding failure in the RS decoder. Then, the attacker focuses on a single inner RM block which they aim to recover. Within this RM block the attacker introduces errors until a decoding failure occurs. Then, they flip each bit to test if it results in a decoding success again. When this happens, the bit was an error-bit with high probability. Since all added vectors are recorded, this method enables a highly probable prediction of a specific bit in the secret y. The strategy comes at the cost of high online query complexity, as each test of the decryption status requires a separate call to the oracle or the PKE.Decrypt procedure with the secret key.

3.3.2 The Two-Phase Attack by Huang et al.

Huang et al. [36] introduced a two-phase key-recovery attack that employs the PC oracle. The attack comprises both online and offline stages. During the online stage, the attacker's objective is to identify a specially crafted, invalid ciphertext $c = (u, v) = (1, v)$ such that adding the secret key's y results in a decoding failure: $\mathcal{C}.\text{Decode}(v + y) \neq \mathcal{C}.\text{Decode}(v)$. The attacker repeated a certain number of attempts to find such a ciphertext. Upon finding a potential candidate, the attacker engaged in educated guessing to discern if a weight-1 pattern explains the decoding behavior. To enhance the likelihood of identifying the correct pattern, the attacker can repeat this procedure. If numerous attempts yield no such ciphertext c, the conclusion is that the inner block has a weight of 0, signifying an all-zero vector block. Thus, this method can determine most of the inner blocks with a bounded weight of 1.

Similarly, the attacker can employ this method to recover the inner blocks of x with weight bounded by 1. Due to the sparsity of the key, a significant number of blocks have weight bounded by 1. Thus, about 50% of the secret positions are recovered. With the public key relation that $x + h \cdot y = s$, the full secret key can be recovered by Gaussian elimination or slight post-processing such as information set decoding.

The primary advantage of this method over the key recovery method from [28] is that when performing educated guess for the value of one inner block of y (or x), only the public decoder of the employed RS RM codes is used, which can be done offline without interaction with the oracle. Thus, the online query complexity can be much lower.

3.3.3 The SCA-LDPC Framework

SCA-LDPC [32] is a generic framework grounded in coding theory for key-recovery chosen-ciphertext side-channel attacks on lattice-based and code-based KEMs using a PC-like oracle. When applied to HQC, this framework significantly reduces query complexity compared to the attacks outlined in [28] and [36].

In contrast to the other attacks, the approach in [32] generates a low-Hamming-weight vector h_l and sets $r_1 = h_l$, while keeping r_2 and e as zero vectors. Consequently, the decryption function takes $v - uy = m\mathbf{G} - h_l \cdot y$ as its input. Unlike the method in [28], which inefficiently recovers only a single y-vector entry per PC oracle call, the SCA-LDPC attack yields more valuable information. This is because each entry in $h_l \cdot y$ is much closer to a uniformly distributed bit, increasing the amount of information gained from each oracle interaction. Further, due to the low-Hamming-weight characteristic of h_l, we can generate a Low-Density Parity-Check (LDPC) code and deploy iterative decoding techniques to effectively recover the sparse secret vector y.

It is worth emphasizing that the attacks from both [36] and [32] leverage the extreme sparsity of HQC's secret key vectors. However, they are based on divergent design principles, making it nontrivial to combine these two attack strategies as per current literature.

3.4 New Method: Zero Testers with Shifting

We next introduce a novel method that tests whether n_2 consecutive bits have weight zero with greater query efficiency. While this approach aligns with the attack described in [36], it diverges in its methods and objectives. Rather than identifying patterns with a bounded weight of 1, our method focuses on efficiently determining blocks with weight zero. We generalize the definition of a block to allow for cyclic shifts of a vector.

Definition 3.1 (Block)**.** A block is a collection of n_2 consecutive bits. Typically blocks are a slice of an element of $\mathcal{R}$. The block may wrap around at the edges — e.g. the last bit, and the first $n_2 - 1$ bits of $x \in \mathcal{R}$ may form a block.

Definition 3.2 (Zero-Block)**.** A zero-block is a block that has zero Hamming weight, i.e. all bits are zero.

The zero-blocks we identify may be at any offset in parts of the secret key's x and y. We are not limited by the RM code's codeword offsets, as we can shift the secret key's components using u and r_2. Specifically, we can craft a ciphertext such that the victim's decoder will correct an error corresponding to shifted versions of x or y.

To construct a ciphertext that contains y shifted by k positions as its error, we can set $r_2 = 0$ and $u = X^k$ and compute v honestly as per the scheme by $v = m\mathbf{G} + s \cdot r_2 + e$. During decapsulation, this will result in $\mathcal{C}$.Decode$(v - u \cdot y)$, where

$$v - u \cdot y = m\mathbf{G} + s \cdot r_2 + e - X^k \cdot y = m\mathbf{G} + e - X^k \cdot y.$$

The error e is then used by the zero-tester method to ascertain information about the first block of this shifted version of y. Analogously, for x, one may set $r_2 = X^k$ and $u = X^k \cdot h$, which results in the computation of

$$v - u \cdot y = m\mathbf{G} + s \cdot X^k + e - X^k \cdot h \cdot y = m\mathbf{G} + x \cdot X^k + e,$$

given that $s = x + h \cdot y$. Both cases result in shifted versions of the secret key becoming the error that the victim's decoder needs to correct. This shifting idea was already introduced in [36]; however, it is made viable through our new zero-testers and we fully develop and implement the idea.

A zero-tester can be used to identify whether a block has weight zero. Zero-testers are errors we add to the ciphertext, such that a decoding failure indicates whether the tested block in the original error of ciphertext can be a zero-block or not — i.e. it indicates whether a block of the secret key has weight zero. To implement this idea we find a sequence of errors (testers) that can be added to the ciphertext's y. The errors e_i are chosen such that adding one or more error bits to them results in a RM decoding failure. This allows efficient testing of whether a block of n_2-bits of the secret key is zero. Using these testers we can perform the attack. The requirement for a single error to detect all possible errors is unsatisfiable since the additional errors could unflip error bits set by the error e. A more reasonable requirement is that each tester should detect a large subset of possible errors induced by the secret key. Since the secret key is sparse the weight of the n_2-bit blocks is typically ≤ 5. Ideally, the union of the sets of secret key errors that the testers detect makes up the vast majority of possible errors that we are likely to encounter. Furthermore, we would like to find the shortest sequence of testers that allows us to detect the whole set of possible errors.

The number of oracle calls our attack requires is highly sensitive to the number of tests required per block. In contrast to other methods, we do not need to search for set-bits - with our zero-testers we can recover n_2 bits with very few queries. Each test requires a single oracle call. For zero-blocks all tests (3 for HQC-128) must be performed to confirm that it does indeed contain no set bits. Since the majority of blocks have a weight larger than 0, we can filter these out early. Only approx. 40% of blocks pass the first stage of the zero-tester in HQC-128. This means for the majority of blocks only a single oracle call needs to be performed. The second stage again only lets 31% of the remaining blocks pass. Thus, even though we use 3 testers for HQC-128, only 1 oracle call is performed per shift for the majority of cases.

The full attack then consists of shifting the respective part of the secret key into a position and performing zero-tests on the first block of the ciphertext. The core procedure of our attack is represented as pseudocode in Figure 2.

3.4.1 Finding Suitable Zero-Testers

Our method of finding a sequence of testers is akin to a mutation fuzzer. We start with random testers of a weight $w_{ini} \in \{\lfloor n_2 \cdot 0.4 \rceil, \ldots, \lfloor n_2 \cdot 0.5 \rceil\}$. Then we perform many mutations on these testers and select the mutation that performs the best. Experimentally we found that mutations that flip 1 or 2 bits yield the best results.

Our evaluation criterion is also probabilistic: we sample a given number of low-weight errors according to the probability distribution of their occurrence in the secret key. This probability distribution can be approximated well using the coefficients of the polynomial given by f:

$$f(x) = ((1-p) + p \cdot x)^{n_2} \tag{4}$$

where $p = \frac{\omega}{n}$. The j^{th} coefficient of f yields the probability that j bits are set in an array of n_2 bits where each bit is i.i.d. with probability p of being set.

After applying a mutation we estimate the success probability of an attack as the probability that the sequence of testers will not fail for the number of queries required by the attack (e.g. 1000). If this estimated attack success probability is greater than 99% we terminate.

If we see no improvement in some iterations of the mutation finding process, we restart the process entirely with newly chosen random errors. This prevents remaining stuck in local optima.

3.5 Selecting Suitable Shifts

During the attack, we shift the secret key to test the weight of different blocks of the key. This raises the question of which of all possible n shifts should be tested first to increase the efficiency of the attack.

3.5.1 Static Shift Selection

A simple approach is to use a fixed sequence of shifts every time we perform the attack. For HQC-128 we used the shifts 0, 192, 92, 297, 238, and 115 plus an offset of bn_2 for the current block $b \in \{0, \ldots, n_1 - 1\}$. These shifts were chosen by greedily picking a shift that performed well in simulation, under the assumption that the previous shifts have already been used to test each block. If these shifts don't suffice we pick random new shifts that have not been tested before.

3.5.2 Dynamic Shift Selection

When performing the attack we obtain new information about the key with each query. Based on the queries performed and their observed responses, we can attempt to search for a shift that will maximize the information gained. One such method could be derived from belief propagation, as the problem shares a similar structure. Once we have an algorithm for estimating the posterior conditional probability distribution of a zero-test for a shift s given the query response pairs observed so far, we could combine it with game tree search algorithms, e.g. a variant of expectimax. This may yield further improved shifts. However, this would come at a significant computational cost, as the branching factor is $n \geq 17669$.

3.5.3 Recoverable Keys and Success Chance

Our attack depends on that there exist sufficiently many zero-blocks in the two parts of the secret key x and y such that we can recover slightly more than 50% of the secret key. This means there may be some keys that cannot be recovered by our attack, since some keys may not have enough zero-blocks. In the simulation we show that for HQC-128 and HQC-256 this is not an issue, and we can recover 98% and 99% of keys respectively, assuming we try all possible shifts exhaustively. For HQC-192 we can only recover approx. 15% of keys, but a large portion of the key can be recovered and be used to reduce the complexity of other attacks.

To obtain the full key in post-processing one can solve the linear equations given by the public key relation $s = \begin{bmatrix} I_n & \text{rot}(h) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$. In practice, the resulting $n \times n$ matrix over $\mathbb{F}_2$ is often not invertible. Thus, one needs to recover additional zero-bits until there is an invertible submatrix. We recover 5 additional bits, which yields a probability of approx. 97% that the $n \times (n-5)$ matrix contains an invertible $(n-5) \times (n-5)$ submatrix. This probability is computed as $p(n,m) = \prod_{i=1}^{m}(1 - 2^{i-1-n})$ [21], where $m = n - 5$.

Choosing fewer than five additional bits tends to decrease the probability of finding an invertible matrix significantly. Conversely, selecting slightly more than five bits, such as increasing the count from 5 to 10, does not markedly enhance this probability since 97% is already quite high. Last, the attack performance suffers if a significantly greater number of additional bits need to be recovered.

Algorithm 1: PC-Oracle Key Recovery Attack

Data: h from the public key, a suitable message m and salt, shifts, zero_testers, and scheme parameters n, n_2, δ
Result: bit_status array: zero-bits in the secret key

bit_status$_{i,j} \leftarrow$ "Unknown" $\forall i \in$ ["X", "Y"], $j \in \{0, \ldots, n-1\}$
for $k \in$ shifts, key_part $\in$ ["X", "Y"] **do**
  if $\bigwedge_{j \in \{s-n_2+1,\ldots,s\}}$ bit_status$_{\text{key_part},j}$ = "KnownZero" **then**
    `// we already know each bit in this block to be zero`
    continue
  end
  classification $\leftarrow$ "ZeroBlock"
  for tester $\in$ zero_testers **do**
    set error e with $\delta = \lfloor \frac{d_{\text{RS}}}{2} \rfloor$ RM-code blocks flipped,
     such that the corruption of the first block
     will cause an RS decoder failure.
    set the first block of e to the used tester
    if key_part = "X" **then**
      $r_2 \leftarrow X^k$
      $u \leftarrow X^k \cdot h$
    else if key_part = "Y" **then**
      $r_2 \leftarrow 0$
      $u \leftarrow X^k$
    end
    construct ciphertext c as $(u, m\mathbf{G} + s \cdot r_2 + e, \text{salt})$
    if $\neg \mathcal{O}_{HQC}(c)$ **then**
      classification $\leftarrow$ "NonZeroBlock"
      break
    end
  end
  if classification = "ZeroBlock" **then**
    for $j \in \{s - n_2 + 1, \ldots, s\}$ **do**
      bit_status$_{\text{key_part},j} \leftarrow$ "KnownZero"
    end
  end
  if $\left(\sum_{i \in [X,Y]} \sum_{j \in \{0,\ldots,n-1\}} \text{bit\_status}_{i,j} = \text{"KnownZero"}\right) \geq n + 5$ **then**
    return bit_status
  end
end
return bit_status

Figure 2: The core of our attack procedure on HQC represented as pseudocode. For brevity we assume that array accesses into bit_status wrap around modulo n.

4 Division Timing Side-Channel

In [28] the authors reveal a timing-side channel stemming from the use of variable-time constant-weight vector sampling. The constant-weight vector sampling is performed as part of the re-encryption step to sample the vectors r_1, r_2, and e. The re-encryption step is part of the decapsulation, as shown in Figure 3. The identified vulnerability was that the sampling process used rejection sampling, which is non-constant-time in the used randomness. The vector sampling process must be constant-time in the used randomness because the randomness depends on the seed θ, which is derived from secret inputs (the message). Leaking any timing information about θ leads to a distinguisher which can be used as a PC oracle. Such a PC oracle completely breaks the cryptosystem leading to full

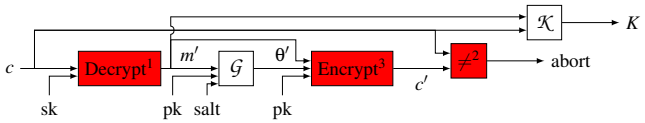

Figure 3: Simplified illustration of HQC decapsulation. Parts with previously identified timing side-channel vulnerabilities are marked in red. Our identified side-channel is in the vector-sampling part of the re-encryption, similar to [28]. It was introduced in the patch to [28]. [1] [46, 60]; [2] [30]; [3] [28].

```
for (size_t i = 0; i < weight; ++i) {
    tmp[i] = i + rand_u32[i] % (PARAM_N - i);
}
```

Figure 4: Vulnerable code snippet generating `div` instructions.

key-recovery.

To remedy the timing side-channel the authors of the scheme implemented a constant-time constant-weight vector sampling algorithm described by Sendrier [54]. This algorithm requires modulo reductions of random numbers, which were derived from the seed θ. Unfortunately, the compiler (gcc v13.1.1) uses division instructions to implement these modulo reductions. Division instructions typically take a variable number of cycles on a CPU, depending on the size of the operands. (cf. Section 4.1). The vulnerable code snippet may be found in Figure 4. We identified the vulnerability through manual analysis of the code's disassembly.

In principle modern compilers have the ability to convert modulo reductions modulo a compile-time known constant into constant-time code using Barrett reductions. This optimization typically occurs when there are single modulo or division operations, that are not within a loop. However, in this case, the compiler does not do so, even when using the `-funroll-all-loops` flag. This flag causes the loop to be unrolled, but the compiler still generates division instructions instead of Barrett reductions.

The timing variation caused by the divisions may be exacerbated if the target system is embedded, and has worse division performance than high-performance x86 microarchitectures. For our analyses and exploitation, we focus on high-performance x86_64 systems, as the vulnerable optimized HQC implementation is written for processors supporting the AVX2 vector instruction-set extension.

4.1 Analyzing Numerator Dependent Division Throughput

To characterize the side-channel leakage that we want to exploit we perform experiments regarding the division throughput on different x86_64 processor microarchitectures.

We measure the throughput of an instruction sequence

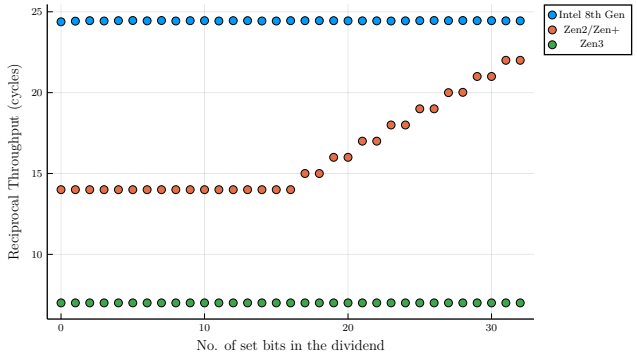

Figure 5: Empirical results of numerator dependent throughput of divisions on different microarchitectures for a fixed divisor (17669). Some high-performance x86 microarchitectures can perform 32-bit divisions by the chosen fixed divisor with numerator-independent throughput, while Zen2 and Zen+ have varying division throughput. Note that we only show the throughput for numerators up to 32-bit and for a fixed divisor, as this is the case most relevant for exploitation of the vulnerability in HQC. We can observe a timing variation on all of the tested architectures if we instead inspect up to 64-bit numerators.

containing a division using nanoBench [3]. The measured instruction sequence is shown in Figure 6.

```
mov r10, 17669 ; divisor
mov rax, numerator
xor rdx, rdx
idiv r10
```

Figure 6: Measured instruction sequence, where numerator is replaced with a concrete numerator.

The instruction sequence was measured on Zen3 (AMD Ryzen 9 5900X), Zen2 (AMD Ryzen 9 3900X), Zen+ (AMD Ryzen 7 3700U) and Intel 8th Gen (Intel i7-8550U). The measured throughput includes not only the division itself but also the preceding instructions. On AMD architectures the `076.00 LsNotHaltedCyc` performance counter was used. On Intel we used `3C.00 CORE_CYCLES`. The benchmark results can be reproduced using the code artifact.

The AMD and Intel optimization manuals are relatively sparse on details regarding latency and throughput of divisions. The AMD optimization manual for the Family 19h line of processors (Zen3, Zen3+, and Zen4) states that: "The hardware integer divider unit has a typical latency of 8 cycles plus 1 cycle for every 9 bits of quotient. The divider allows limited overlap between two consecutive independent divide operations. 'Typical' 64-bit divides allow a throughput of one divide per 8 cycles (where the actual throughput is data dependent)" [11]. For the Family 17h (Zen, Zen+, and Zen2) the

optimization manual states: “The radix-4 hardware integer divider unit can compute 2 bits of results per cycle” [10].

To exploit the variable division runtime we target a Zen2 machine. We choose Zen2 due to the variable division timing even for 32-bit numerators.

Using just timing information on the vector sampling part of the HQC decapsulation on an otherwise idle machine, we are able to distinguish a fast vector sampling from a random one. We obtain an accuracy of up to 80% (both classes represented equally). However, when measuring the execution time of the entire decapsulation, the timing signal becomes too weak for practical exploitation. Even 100 000 timing measurements do not suffice to gain any significant distinguishing advantage.

4.2 DIV-SMT

We introduce DIV-SMT, which enables us to precisely measure only the contribution of the `div` instructions to the timing signal, and filter out noise caused by the other parts of the cryptosystem - less than 2% of the decapsulation runtime is dedicated to the divisions relevant to our attack. Like prior works [8, 25], we use contention among SMT sibling threads to detect the execution of a certain instruction (`div`). In addition and in contrast to these works, we also use contention to infer information about the processed operands (the number of leading zeros in the numerators).

4.2.1 Threat Model

DIV-SMT is relevant to all contexts where the attacker can execute code co-located to the victim. Crucially, the CPU must support SMT and have it enabled. Disabling SMT comes with a typically high but workload-dependent cost [42, 51]. Especially in systems with high SMT thread count like Power10, which sports up to 8 SMT threads per core, it is very desirable to leave SMT enabled. Further, the attacker must not be impeded by scheduling countermeasures such as Linux’s Core Scheduling [1]. Core Scheduling may prevent the attacker from running on the same core as the victim, but also comes with a performance cost. In certain cases, the performance may be worse than with SMT disabled due to the additional scheduling overhead [20].

4.2.2 Side-Channel Measurement

The setup for DIV-SMT is as follows: the attacker runs a thread pinned to the sibling SMT thread of the victim’s thread. For our attack simulation, we also pin the victim’s thread. In more restricted scenarios, the attacker may not have permissions to pin other user’s processes. To counteract this, the attacker may start many DIV-SMT receivers [24]. If possible, these receivers may be pinned to a different physical core to reduce measurement noise.

```
fn rdpru32() -> u32 {
    let lo;
    unsafe {
        asm!("rdpru",
            out("eax") lo,
            out("edx") _,
            in("rcx") 1,
        );
    }
    lo
}

let mut i = 0;
while !done_decaps.load(Ordering::SeqCst) && i < len {
    measurements[i] = rdpru32();
    let mut _dividend = 1;
    let mut _remainder: u64 = 0;
    let divisor = 1;
    unsafe {
        asm!(
            ".rept 5",
            "xor edx, edx",
            "div rcx",
            ".endr",
            inout("rax") _dividend,
            inout("rdx") _remainder,
            in("rcx") divisor,
        );
    }
    i += 1;
}
```

Figure 7: Excerpt from the attacking code showing the measurement loop and use of the `rdpru` instruction. In the loop body the `APERF` performance-counter is stored into the measurement array and 5 divisions are executed. The loop continues until either the maximum trace length is reached (in which case the trace is discarded), or the decapsulation has finished.

The attacker executes division instructions in a loop, measures the cycle counter and stores the counter for measurement in an array. For the cycle counter measurement, we use the `rdpru` instruction. The `rdpru` instruction allows us to read the Actual Performance Frequency Clock Counter (`APERF`) performance-counter from user-space which counts the number of cycles executed by the specific core the executing thread is running on [43]. This is opposed to the `rdtsc` instruction which reads a globaö core-frequency-independent timestamp counter. Further, in our experiments, `rdpru` executes faster than `rdtsc` which enables higher temporal resolution in our SMT side-channel traces. An excerpt from the attacking loop code may be found in Figure 7.

When the victim is executing divisions, the execution unit for divisions is busy, and the attacker’s divisions are also slowed down. Slowing the attacker’s divisions down causes an increase in the cycle delta between measurements for the attacker. When the victim does not use the execution unit for divisions, the attacker’s divisions complete at a faster rate

enabling us to filter out parts of the cryptosystem that do not use divisions and are therefore irrelevant to our attack. We use an empirically determined constant threshold to classify whether the sibling SMT thread is performing divisions. We then sum up these division timings to obtain a total division runtime for a single trace. After collecting 100 traces and computing each trace's total division runtime, we compute the median total division runtime. This division runtime is then thresholded to form a classifier.

In principle, we use the contention timing side-channel for two distinct purposes. Similar to e.g. PortSmash [8] and SQUIP [25]), we use the contention side-channel to detect the execution of division instructions. Additionally, and unlike them, we use the side-channel to infer information about the processed operands, an aspect of SMT contention attacks, which, to the best of our knowledge, has not been exploited before.

4.2.3 Practical Exploitation in HQC

For our HQC attack, we compute a threshold using ciphertexts where it is known whether a decoding failure occurs. Such ciphertexts are easy to construct by setting the error to a large or small value. The classifier then determines whether the total division runtime is faster than a random message or not. If it is faster, we know that decoding succeeded.

With the SMT based side-channel, we obtain above 90% accuracy with 100 traces. To further increase the accuracy, we measure the true-positive-rate and true-negative-rate of the classifier, and make a decision based on multiple classifications by computing the probability of a positive or negative class, given the observed classifications. A majority vote could also be used and would likely perform similarly.

Finding suitable zero testers is the most costly part of the attack, which is why we would like to only perform it once. By fixing a single message first we can achieve that goal, since the zero testers depend solely on the first RM codeword.

In the round 4 submission of HQC the public key and salt are added as inputs to derive the seed θ, which in turn determines the generated numerators. The addition of the salt allows us to brute-force a salt, that, together with the fixed message and a given public key, generates suitable numerators. This process is also illustrated in Figure 8. In general, we want to find a salt that minimizes the division runtime of the generated numerators as this will increase the distinguishing capability. In our experiments we brute-force salts for each attack such that we expect a timing difference of at least 55 cycles from a median ciphertext's numerators total division runtime.

4.2.4 Evaluating the Side-Channel

We evaluate how many traces are necessary to classify a ciphertext with a desired accuracy. In Figure 9 we show that

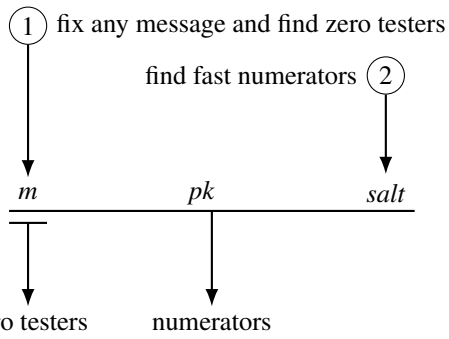

Figure 8: We first choose an arbitrary message for which we intend to find suitable zero testers. Once we've found suitable zero testers for this message we can attack any public key by varying the salt. By changing the salt we change which numerators are generated. These are used during the re-encryption step and leak timing information about the decoded message. We need to distinguish our chosen numerators from random different ones. Therefore, we want to find a salt that generates fast divisions with small numerators.

within our lab environment the number of traces per oracle call could still be halved without a significant drop in the classification accuracy. However, the number of traces required in other noise contexts may deviate significantly.

In Figure 10 we demonstrate the clear difference in the timing distributions of fast and random ciphertexts as observed by the DIV-SMT side-channel.

In Figure 11 we show that the computed division timing of a ciphertext strongly correlates with the timing observed by the DIV-SMT side-channel. For Zen2 the number of cycles per division is estimated as $8 + \lfloor(\lfloor\log_2(q+1)\rfloor + 1)/2\rfloor$, where q is the resulting quotient of the division. This model is derived from the optimization guide [10].

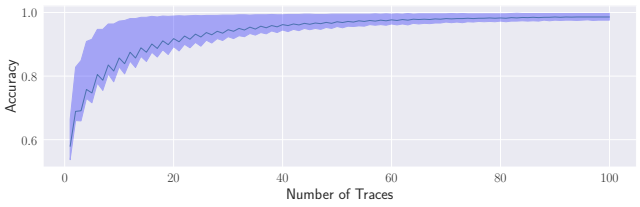

Figure 9: Oracle accuracy over the number of traces used to determine the median division timing. The line represents the median oracle accuracy, while the shaded region represents the 5th and 95th percentile.

5 Experimental Results

In this section, we present the experimental results of our research, organized into two distinct subsections. We begin by

	$O_{HQC}^{perfect}$	O_{HQC}^{iedal}	$O_{HQC}^{0.995}$	$O_{HQC}^{0.95}$	$O_{HQC}^{0.9}$	$O_{HQC}^{0.515}$
SCA-LDPC [32]	9 000	10 000	18 000	35 250	59 500	n/a
Our Work	1 094	1 142	2 246	4 922	6 951	5 728 728
Success Rate	78.00%	77.30%	76.90%	77.10%	76.60%	77.13%

Table 3: Comparison of median oracle calls required in our attack vs. SCA-LDPC. For our attack we additionally show the success rate. Attacks are classified as successful if they managed to recover $n+5$ zero bits from the secret key and no bits were incorrectly identified as a zero bit. These $n+5$ zero bits are sufficient in approx. 97% of cases as discussed in Section 3.5.3.

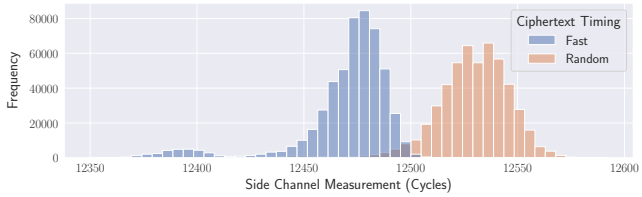

Figure 10: Side-channel measurement for fast and random ciphertexts. Here, fast ciphertexts with a computed division timing deviation of 55 cycles from the median division timing were selected.

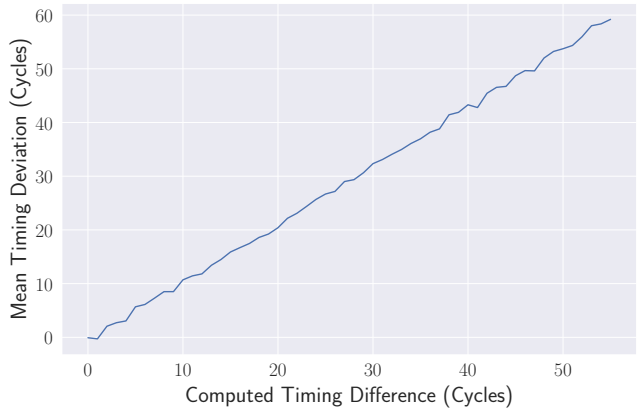

Figure 11: Mean measured difference between the division timing of fast and random messages as measured by the side-channel over the estimated division timing deviation.

showing the simulation results, which illustrate the enhanced performance of our method across a range of oracle accuracies. Subsequently, we present real-world attack results on HQC-128 executed on an AMD Zen2 platform, demonstrating its practical effectiveness and confirming that our simulation findings align with real-world applications.

5.1 Attack Simulation

To provide evidence that our new attack strategy works, we perform attacks using a simulated side-channel oracle. The simulated oracle obtains the true information that is to be leaked and adds oracle accuracy dependent noise to it. The true information in our case is whether the decoder outputs the original message or not. We simulate the attack in 6 different scenarios: perfect, ideal, 0.995, 0.95, 0.9 and 0.515, denoted by $O_{HQC}^{perfect}$, O_{HQC}^{ideal}, $O_{HQC}^{0.995}$, $O_{HQC}^{0.95}$, $O_{HQC}^{0.9}$ and $O_{HQC}^{0.515}$, respectively.

The perfect side-channel scenario is unattainable in practice; this oracle simply reveals the true information—whether the decoder outputs the original message or not. Nonetheless, this concept plays a crucial role in the misuse or key-mismatch attack scenarios [15].

The ideal scenario is relevant to the round 3 implementation of HQC – it is similar to the perfect oracle, except that a message-dependent deterministic failure rate of 0.0058 for decoding failures is simulated. Specifically, when the decoder outputs a message m' different from the original message m there's a chance of 0.0058 that the ideal oracle will output that $m = m'$, even though they are different. This simulates the case where the message m' has the same timing behavior as the message m and is therefore indistinguishable. We use this oracle for the sake of comparison to previous works, as in its current form it no longer applies to the HQC round 4 implementation, since the timing vulnerability discovered in [28] has been patched.

The other oracles build on the ideal oracle and add independently sampled random noise: for an accuracy level of $\rho = 0.995$ there is a $1-\rho = 0.005$ chance that the result will be flipped.

In Figure 12, we show the number of oracle calls the attack requires to complete an attack under different noise conditions. The oracle here is simulated and uses a random number generator to simulate the random failures during a real-world side-channel attack. It can be seen that the distribution has a long tail, which consists of attacks that fail because they cannot find a sufficient number of zero-blocks.

In Table 3, we compare the median number of oracle calls required by our attack versus the SCA-LDPC framework. Our findings indicate a substantial improvement with the new Zero Tester method, which necessitates only 11% to 13% of the oracle calls compared to the SCA-LDPC framework. Notably, this ratio of efficiency gains remains consistent even as the oracle's accuracy decreases to 0.9.

We simulate the performance of the PC oracle at the accu-

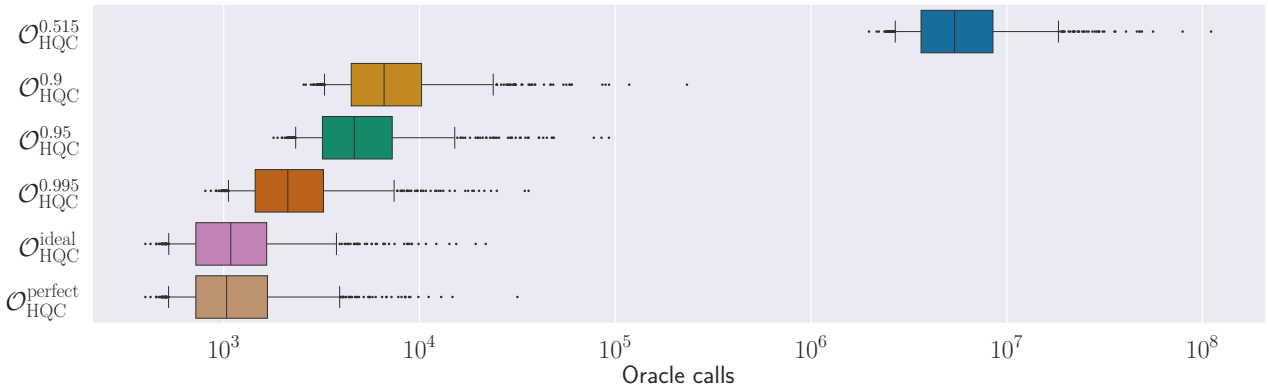

Figure 12: Oracle call requirements for attacking HQC-128 under various oracle accuracies, with whiskers representing the 5th and 95th percentile. We performed 1 000 simulated attacks per oracle accuracy.

racy of 0.515. As noted in [59], this specific accuracy level corresponds to the effectiveness of a Neural Network model in constructing a PC oracle from a masked hardware implementation. For such an oracle $\mathcal{O}_{\mathrm{HQC}}^{0.515}$, estimating the number of oracle calls required by the SCA-LDPC framework is resource-intensive, a task not undertaken in [32]. In contrast, our simulations demonstrate a median requirement of only 5 728 728 oracle calls.

If desired one may reduce the number of oracle calls further by trading it for computation time. Although the trade-off comes with quickly diminishing returns as Information-Set Decoding (ISD) [47,56] comes with high costs. In Table 3, we simulate a success probability ranging between 76% and 78%. Enhancing the success rate is achievable through employing more resource-intensive ISD for post-processing, as opposed to Gaussian Elimination.

5.2 Real-World Attacks in an AMD Zen2 Platform

We performed 1000 attacks using our DIV-SMT side-channel oracle. The targeted machine is a Zen2 machine featuring an AMD Ryzen 7 3700X 8-Core Processor. The attacks ran concurrently on 8 cores, each using 2 SMT threads. 75.3% of the attacks succeeded in recovering $n+5$ zero-bits from the secret key. This success rate comes close to the 76% to 78% success rate that we observe in simulation (cf. Table 3). A median of 465 409 DIV-SMT traces were used to form responses to a median of 2 577 oracle calls in a median attack runtime of 111 seconds. The number of traces includes a median number of 204 180 calibration traces. Using the calibrated threshold the SMT oracle was tested to have an accuracy of 98.8%.

6 Discussions

In this section, we discuss the discovered vulnerability and attack methodology from various perspectives. We first address countermeasures against the DaS vulnerability in HQC. Subsequently, we compare our new key-recovery method to the state-of-the-art SCA-LDPC framework, elucidating the sources of performance improvement. Lastly, we explore the potential limitations of the new Zero Tester method.

6.1 Countermeasures

In addressing the DaS vulnerability within HQC, two main countermeasures emerge:

1. Manually code the necessary Barrett reductions.
2. Use multiplication and bit-shift as in Bit Flipping Key Encapsulation (BIKE)

The HQC design team and the PQClean implementers, after our public disclosure [9], chose the first approach – it stays true to the specification of the scheme. Barrett reductions are a way to compute $a \bmod n$ when n is a known constant in constant-time (the timing is independent of a). Specifically, one computes the quotient $q = a/n$ and then the remainder as $a - qn$. The quotient q is approximated by $q = \lfloor (a+1)m/2^k \rfloor$ for a suitable k and m. Here m is computed as $\lfloor 2^k/n \rfloor$, such that $m/2^k$ is an approximation of $1/n$ and $a+1$ is used instead of a to prevent underestimating the quotient. The floored division by 2^k can be implemented through a bit-shift, which is constant-time on most systems, especially in the shifted operand. For further details we refer to the updated HQC optimized reference implementation.

BIKE replaces the modulo reduction with rounding: Instead of $i + (s_i \mod n - i)$ they compute $i + \lfloor (n-i)s_i/2^{32} \rfloor$. The latter function can be implemented using an integer multiplication and a bit-shift. This function generates the same noticeable bias as the modulo reduction when compared to a uniform distribution. Note that the bias is not a security issue [54].

Variants	P_0	P_1	P_2	P_3	P_4	$P_{\geq 5}$
HQC-128	23.44%	34.38%	24.83%	11.77%	4.12%	1.45%
HQC-192	16.50%	30.00%	27.00%	16.04%	7.07%	3.40%
HQC-256	23.14%	34.06%	24.87%	12.02%	4.32%	1.59%

Table 4: Approximation of the probability that a randomly sampled inner block in the secret vector has a certain Hamming weight [27]. P_i denotes the probability that the block has a Hamming weight of i.

When possible we recommend to use the method used by BIKE as it is easier to implement (especially cross-platform), less error-prone, and most likely faster on most target architectures. We recommend a change in the specification in any case: implementers of the modulo reduction in the specification are likely to make the same mistakes unless additional guidance is given.

6.2 Comparison to SCA-LDPC

The SCA-LDPC framework [32] is a generic framework that can apply not only to code-based schemes but also to lattice-based schemes with a larger alphabet and a denser secret distribution. Our new PC key-recovery method is tied to HQC and exploits the extreme sparsity of the secret vector and the concatenated code construction used in HQC. The latter allows us to check whether blocks of consecutive bits in the secret key have weight 0, and due to the low weight of the secret key, this is often the case.

Our method provides substantial improvements over the SCA-LDPC framework for HQC, mainly for two reasons. Firstly, our approach, which focuses on identifying n zero positions from the $2n$ unknowns, requires extracting less mutual information compared with the SCA-LDPC framework that aims to recover the entire secret vector y or x. Considering the HQC-128 parameter set, the latter must uniquely determine y or x among $\binom{n}{\omega} \approx 2^{623}$ possible configurations. In contrast, by exploiting public key information, our method needs to recover only approx. 129 bits of information. This is roughly estimated by subtracting the entropy associated with a binary vector of dimension n and weight 2ω from the entropy of two binary vectors, each of dimension n and weight ω:

$$\binom{n}{w}^2 / \binom{n}{2w} \approx 2^{129}. \tag{5}$$

Secondly, our method has the potential to extract considerably more information from a single side-channel trace. For instance, in the HQC-128 context with a perfect oracle, the SCA-LDPC framework requires about 9000 oracle calls (cf. Table 3) to gain 623 bits of information. In contrast, simulations indicate that our approach, in its first pass to test the key's 92 RM blocks, expects to recover approximately 8300 zero positions with only 172 oracle calls. This could reduce the number of possible patterns by a factor of approx. 2^{47}.

6.3 Limitations

Lastly, we discuss the limitations of the new Zero Tester method and the potential for improvement. The efficacy of our approach is contingent upon finding long sequences of all-zero entries between two ones in the secret polynomials x and y. The probability that an inner RM block within the secret vector is entirely zero is over 23% for HQC-128 and HQC-256, decreasing to 16.5% for HQC-192, as shown in Table 4. This reduction in likelihood explains the limited applicability of the new method to HQC-192, where it is effective for only about 15% of keys, as discussed in Section 3.5.3. For the remaining keys, the positions recovered are not adequate to achieve full-key recovery using Gaussian Elimination. Resorting to the more computationally demanding ISD algorithms for post-processing could enable the recovery of a greater number of keys. We defer a detailed quantitative analysis of this aspect to future research.

7 Conclusion

In this work, we have identified a novel timing side-channel vulnerability, an instance of the DaS vulnerability class, within the optimized, supposedly constant-time, reference implementation of the HQC round 4 submission to NIST. This vulnerability, characterized by its generic nature as demonstrated by similar issues in cryptographic implementations like KyberSlash, poses a significant security risk. Our proposed DIV-SMT methodology effectively targets this vulnerability on SMT-enabled processors by leveraging contention among SMT sibling threads. This process not only identifies specific instructions but also deduces operand information, thereby establishing a PC oracle distinguisher that enables a key-recovery timing attack.

Additionally, we have introduced a new key recovery approach for HQC leveraging the PC oracle, which significantly lowers the requirement for side-channel traces. Through comprehensive simulations using a simulated PC oracle at various levels of accuracy, coupled with practical attacks executed on an AMD Zen2 CPU, our research confirms both the effectiveness and practical viability of our novel attack strategy.

Acknowledgements

We would like to thank the reviewers and our shepherd for their valuable feedback.

This research was supported in part by the National Research Center for Applied Cybersecurity ATHENE as part of the PORTUNUS project in the research area Crypto, the Austrian Science Fund (FWF SFB project SPyCoDe 10.55776/F85), the Swedish Research Council (grant numbers 2021-04602 and 2023-03654), the Swedish Civil Contingencies Agency (grant number 2020-11632), the Swedish Foundation for Strategic Research (grant number RIT17-0005) and the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation. Additional funding was provided by a generous gift from Google. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

References

[1] Core scheduling. https://www.kernel.org/doc/html/latest/admin-guide/hw-vuln/core-scheduling.html.

[2] Ibm power e1080 technical overview and introduction. https://www.redbooks.ibm.com/redpapers/pdfs/redp5649.pdf.

[3] Andreas Abel and Jan Reineke. nanoBench: A Low-Overhead Tool for Running Microbenchmarks on x86 Systems. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 34–46. IEEE, 2020.

[4] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jerome Lacan, Jean-Marc Robert, and Pascal Veron. HQC. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

[5] Carlos Aguilar-Melchor, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Efficient encryption from random quasi-cyclic codes. *IEEE Transactions on Information Theory*, 64(5):3927–3943, 2018.

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

[7] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions.

[8] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri. Port Contention for Fun and Profit. In *S&P*, 2018.

[9] ambiso. Hqc implementation out of date/vulnerable, 2023. https://github.com/PQClean/PQClean/issues/482.

[10] AMD. Software Optimization Guide for AMD EPYC 7002 Processors, 3 2020. https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/software-optimization-guides/56305.zip.

[11] AMD. Software Optimization Guide for AMD EPYC 7003 Processors, 11 2020. https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/software-optimization-guides/56665.zip.

[12] AMD. Software Optimization Guide for the AMD Zen4 Microarchitecture, 1 2023. https://www.amd.com/content/dam/amd/en/documents/processor-tech-docs/software-optimization-guides/57647.zip.

[13] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, Valentin Vasseur, Santosh Ghosh, and Jan Richter-Brokmann. BIKE. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

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

[15] Ciprian Băetu, F. Betül Durak, Loïs Huguenin-Dumittan, Abdullah Talayhan, and Serge Vaudenay. Misuse attacks on post-quantum cryptosystems. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 747–776. Springer, Heidelberg, May 2019.

[16] Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 311–323. Springer, Heidelberg, August 1987.

[17] Daniel J. Bernstein. variable-time kyber ref software, 2023. https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/hWqFJCucuj4/m/-Z-jm_k9AAAJ.

[18] NIST Computer Security Resource Center. Round 4 Submissions - Post-Quantum Cryptography | CSRC, 2023. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions accessed: 2024-01-17.

[19] Tomás Fabsic, Viliam Hromada, Paul Stankovski, Pavol Zajac, Qian Guo, and Thomas Johansson. A reaction attack on the QC-LDPC McEliece cryptosystem. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, pages 51–68. Springer, Heidelberg, 2017.

[20] Dario Faggioli. Re: [RFC PATCH v3 00/16] Core scheduling v3, 2019.

[21] Alfonso Fernandez. Probability that a random binary matrix will have full column rank? Mathematics Stack Exchange. https://math.stackexchange.com/q/564699 (version: 2013-11-12).

[22] Agner Fog. The microarchitecture of Intel, AMD, and VIA CPUs: An optimization guide for assembly programmers and compiler makers, 2021. https://www.agner.org/optimize/microarchitecture.pdf.

[23] Philippe Gaborit and HQC team. Technical update in the proof of hqc, 2024. https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/sRZk_CAHUWU/m/oLX_2ns9BAAJ.

[24] Stefan Gast, Jonas Juffinger, Lukas Maar, Christoph Royer, Andreas Kogler, and Daniel Gruss. Remote scheduler contention attacks. In *Financial Cryptography and Data Security - 28th International Conference, FC 2024, Revised Selected Papers*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer-Verlag, March 2024. Financial Cryptography and Data Security 2024, FC 2024 ; Conference date: 04-03-2024 Through 08-03-2024.

[25] Stefan Gast, Jonas Juffinger, Martin Schwarzl, Gururaj Saileshwar, Andreas Kogler, Simone Franza, Markus Köstl, and Daniel Gruss. SQUIP: Exploiting the Scheduler Queue Contention Side Channel. In *S&P*, 2023.

[26] Franziskus Kiefer Goutam Tamvada, Karthikeyan Bhargavan and Peter Schwabe. Updated poly_tomsg to prevent a compiler from using div, 2023. https://github.com/pq-crystals/kyber/commit/dda29cc63af721981ee2c831cf00822e69be3220.

[27] Guillaume Goy, Antoine Loiseau, and Philippe Gaborit. A new key recovery side-channel attack on HQC with chosen ciphertext. In *Post-Quantum Cryptography: 13th International Workshop, PQCrypto 2022, Virtual Event, September 28–30, 2022, Proceedings*, pages 353–371. Springer, 2022.

[28] Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. Don't reject this: Key-recovery timing attacks due to rejection-sampling in HQC and BIKE. *IACR TCHES*, 2022(3):223–263, 2022.

[29] Qian Guo and Thomas Johansson. A new decryption failure attack against HQC. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 353–382. Springer, Heidelberg, December 2020.

[30] Qian Guo, Thomas Johansson, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 359–386. Springer, Heidelberg, August 2020.

[31] Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 789–815. Springer, Heidelberg, December 2016.

[32] Qian Guo, Denis Nabokov, Alexander Nilsson, and Thomas Johansson. SCA-LDPC: A Code-Based Framework for Key-Recovery Side-Channel Attacks on Post-Quantum Encryption Schemes. *ASIACRYPT 2023*, 2023. https://eprint.iacr.org/2023/294.

[33] Chris Hall, Ian Goldberg, and Bruce Schneier. Reaction attacks against several public-key cryptosystems. In Vijay Varadharajan and Yi Mu, editors, *ICICS 99*, volume 1726 of *LNCS*, pages 2–12. Springer, Heidelberg, November 1999.

[34] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Heidelberg, November 2017.

[35] HQC team. Updated submission package (round 4) (2024/02/23), 2024. https://pqc-hqc.org/download.php?file=hqc-submission_2024-02-23.zip.

[36] Senyang Huang, Rui Qi Sim, Chitchanok Chuengsatiansup, Qian Guo, and Thomas Johansson. Cache-Timing Attack Against HQC. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023, Issue 3:136–163, 2023. https://tches.iacr.org/index.php/TCHES/article/view/10959.

[37] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS$^+$. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[38] Intel. Intel 64 and IA-32 Architectures Optimization Reference Manual, 2023. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html.

[39] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson. SIKE. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

[40] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Heidelberg, August 1996.

[41] Adam Langley. Lucky Thirteen attack on TLS CBC. https://www.imperialviolet.org/2013/02/04/luckythirteen.html, 2013-02-04. accessed 2023-10-02.

[42] Michael Larabel. Intel Hyper Threading Performance With A Core i7 On Ubuntu 18.04 LTS, 6 2018.

[43] Moritz Lipp, Daniel Gruss, and Michael Schwarz. AMD Prefetch Attacks through Power and Time. In *USENIX Security*, 2022.

[44] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[45] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked IND-CCA secure saber KEM implementation. *IACR TCHES*, 2021(4):676–707, 2021. https://tches.iacr.org/index.php/TCHES/article/view/9079.

[46] Thales Bandiera Paiva and Routo Terada. A timing attack on the HQC encryption scheme. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 551–573. Springer, Heidelberg, August 2019.

[47] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[48] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[49] Yue Qin, Chi Cheng, Xiaohan Zhang, Yanbin Pan, Lei Hu, and Jintai Ding. A systematic approach and analysis of key mismatch attacks on lattice-based NIST candidate KEMs. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 92–121. Springer, Heidelberg, December 2021.

[50] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR TCHES*, 2020(3):307–335, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8592.

[51] Yaoping Ruan, Vivek S. Pai, Erich Nahum, and John M. Tracey. Evaluating the Impact of Simultaneous Multithreading on Network Servers Using Real Hardware. In *SIGMETRICS*, 2005.

[52] Thomas Schamberger, Lukas Holzbaur, Julian Renner, Antonia Wachter-Zeh, and Georg Sigl. A Power Side-Channel Attack on the Reed-Muller Reed-Solomon Version of the HQC Cryptosystem. In Jung Hee Cheon and Thomas Johansson, editors, *Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, Virtual Event, September 28-30, 2022, Proceedings*, volume 13512 of *Lecture Notes in Computer Science*, pages 327–352. Springer, 2022.

[53] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[54] Nicolas Sendrier. Secure Sampling of Constant-Weight Words – Application to BIKE. Cryptology ePrint Archive, Paper 2021/1631, 2021. https://eprint.iacr.org/2021/1631.

[55] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994.

[56] Jacques Stern. A method for finding codewords of small weight. In *Coding Theory and Applications: 3rd International Colloquium Toulon, France, November 2–4, 1988 Proceedings 3*, pages 106–113. Springer, 1989.

[57] Jakub Szefer. Survey of microarchitectural side and covert channels, attacks, and defenses. *Journal of Hardware and Systems Security*, 3(3):219–234, 2019.

[58] Mohammadkazem Taram, Xida Ren, Ashish Venkat, and Dean Tullsen. SecSMT: Securing SMT Processors against Contention-Based Covert Channels. In *USENIX Security*, 8 2022.

[59] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/EM analysis on post-quantum KEMs. *IACR TCHES*, 2022(1):296–322, 2022.

[60] Guillaume Wafo-Tapa, Slim Bettaieb, Loic Bidoux, Philippe Gaborit, and Etienne Marcatel. A practicable timing attack against HQC and its countermeasure. Cryptology ePrint Archive, Report 2019/909, 2019. https://eprint.iacr.org/2019/909.

[61] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 679–697. USENIX Association, August 2022.

[62] Keita Xagawa, Akira Ito, Rei Ueno, Junko Takahashi, and Naofumi Homma. Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 33–61. Springer, Heidelberg, December 2021.