PQ-Hammer: End-to-end Key Recovery Attacks on Post-Quantum Cryptography Using Rowhammer

Samy Amer
Georgia Tech
s.amer@gatech.edu

Yingchen Wang
UC Berkeley
yingchenwang@berkeley.edu

Hunter Kippen
UMD and Samsung Research
h.kippen@samsung.com

Thinh Dang
NIST
thinh.dang@nist.gov

Daniel Genkin
Georgia Tech
genkin@gatech.edu

Andrew Kwong
UNC Chapel Hill
andrew@cs.unc.edu

Alexander Nelson
University of Arkansas
ahnelson@uark.edu

Arkady Yerukhimovich
George Washington University
arkady@gwu.edu

Abstract—**As post-quantum cryptography (PQC) nears standardization and eventual deployment, it is increasingly important to understand the security of the implementations of selected schemes. In this paper, we conduct such an investigation, uncovering concerning findings about many of the finalists of the NIST PQC standardization competition.**

Specifically, we show Rowhammer-based attacks on the Kyber and BIKE Key Exchange Mechanisms and the Dilithium Digital Signature scheme that enable complete recovery of the secret key with only a moderate amount of effort – no supercomputers, or months of precomputation. Moreover, we experimentally carry out our attacks using a combination of Rowhammer, performance degradation, and memory massaging techniques, showing that our attacks are practically feasible.

Our results show that such side-channel based attacks are a critical concern and need to be considered when new cryptographic schemes are standardized, when standard implementations are developed, and when instances are deployed. We conclude with recommendations on implementation techniques that harden cryptographic schemes against Rowhammer attacks.

1. Introduction

The combined effect of quantum computers and Shor's algorithm threaten to trivialize the mathematical problems that underlie classical asymmetric key cryptography. To combat this issue, the cryptographic community has been developing quantum-resistant cryptographic primitives. These have been developed into full cryptographic algorithms for key establishment and digital signatures, and are undergoing a number of standardization efforts globally. Collectively, this effort has been given the name "post-quantum cryptography (PQC)". Considerable effort has been taken in cryptanalysis of the PQC primitives, resulting in a selection of schemes that have resisted all known classical and quantum cryptanalytic attacks.

However, with the move towards practical implementations of these algorithms, we must additionally consider the side channel security of these implementations. Side channels pose a consistent challenge to the security of cryptographic implementations due in-part to the sheer size of the problem space of preventing leakage or manipulation that can allow adversaries to extract secret information. NIST has made explicit the need to create side channel resilient implementations of these algorithms [1]–[4].

Side channel attacks can take many forms, but a particularly concerning mechanism exists in the form of the Rowhammer vulnerability [5]. Rowhammer allows a remote attacker to intentionally flip bits in a victim process's memory to cause unintended behaviors. Two recent works have considered Rowhammer vulnerabilities in PQC schemes ("Signature Correction" by Islam et al. [6], and "FrodoFlips" by Fahr et al. [7]), and found that there are mechanisms which can be used to leak secret key information from the Frodo Key Encapsulation Mechanism and the Dilithium Digital Signature schemes respectively.

Using these works as motivation, we seek to further understand the security of PQC schemes against Rowhammer attacks. In particular, we are interested in end-to-end attacks that can fully recover the secret key of the target schemes – prior work only allowed recovering a session key or weakening security of the schemes they targeted.

1.1. Our Contributions

Demonstrating that many PQC schemes allow such key recovery attacks, in this paper we show practical key recovery experiments against the reference implementations of the Kyber and BIKE Key Encapsulation Mechanisms and the Dilithium Digital Signature, all of which are among the finalist schemes selected by the NIST standardization process. The chosen schemes represent multiple fundamentally-different approaches to PQC security with lattice-based techniques used for Kyber and Dilithium and coding-based techniques for BIKE. For these schemes, we achieve the following (see Table 1).

- We exploit a Rowhammer vulnerability in Kyber, demonstrating end-to-end secret key recovery (Section 4).
- We demonstrate an end-to-end secret key recovery on BIKE-KEM through instruction flipping (Section 5).
- We demonstrate a key recovery on deterministic and randomized Dilithium (Section 6).
- Finally, we make recommendations on how to harden PQC implementations against Rowhammer (Section 7).

1.2. Attack Overview

The primary focus of our research is to identify and execute bit flip attacks on candidate or standardized PQC schemes. During our investigation of implementations of these algorithms, we focused on the reference implementations as provided to the standardization efforts. A particular emphasis of this effort was to fully recover the full secret key as opposed to prior literature which obtained only partial key information (SignatureCorrection [6]) or session keys (FrodoFlips [7]). Additionally, we sought to identify key-recovery attacks that would not be detected in current specifications so that any successful attack could have a longer-term impact.

For each scheme, we followed a repeatable pattern to identify potential key recovery attacks. For the two key-encapsulation mechanisms (KEMs), we focused on manipulations to the key generation algorithm that enable secret key recovery attacks. For the signature scheme (Dilithium) we instead focused on the sign function. These two functions were selected because the secret key will be in memory during these functions, and therefore are potentially susceptible to bit-flip attacks.

Once a target function was identified, we inspected the reference code and compared constraints on parameters assumed by the cryptographic specification to the checks performed in the code. In particular, we identified variables where a side-channel attack could induce the variable to violate the assumed conditions of the cryptographic implementations without causing a detectable failure. We were particularly interested in variables whose manipulation could lead to the scheme disclosing its secret key.

Once we identified these variables ("target variables") we created simulations of the attack by directly manipulating the reference implementation code to verify that the attack had the intended effect. Once a simulated attack was verified, we compiled the reference code directly and disassembled the binary to assure that compiler optimizations would not prevent the attack. Each process was dynamically analyzed to determine the amount of time that is available to launch a bit flipping attack. If we found that this amount of time was not sufficient for a Rowhammer attack to successfully flip the necessary bits, we performed performance degradation to slow the victim process during the critical section of code. We launched our Rowhammer attack by activating a hammering process, a degradation process, and the victim process in parallel. Finally, we used the modified process to extract the secret-key information for the target cryptographic scheme. In some cases, e.g., for Kyber and Dilithium, this last step required multiple communication sessions to be established with the victim process to enable key recovery.

1.3. Comparison to Prior Work

We now briefly compare our attacks to those shown in prior work.

Attacks on Key Encapsulation Mechanisms. Our attack on the Kyber protocol is similar to the attack shown against FrodoKEM in FrodoFlips [7]. Both attack the KeyGen function to hammer the secret key during key generation to cause an increased rate of ciphertexts causing a decryption failure. Both attacks then use a number of such failing ciphertexts to recover the secret key. However, there are some critical differences between these attacks that result in a stronger full key recovery attack in the case of Kyber.

In FrodoFlips, the authors' attack required flipping 8 very targeted bits of the secret key, without flipping any of the other bits, during key generation. This challenge in fact caused their attack to fall short of recovering the full secret key as the authors were only able to flip 7 of the 8 bits – enough for a brute-force attack to recover established session keys, but not enough to recover the long-term secret key. Additionally, their attack required generating 665,000 failing ciphertexts (e.g. producing decryption errors), necessitating the use of a super computer to produce sufficiently many ciphertexts to run the attack.

In our attack on Kyber, on the other hand, we only need to flip two bits of the secret key, making the time needed to run the Rowhammer attack much smaller than in the FrodoFlips case. However, it turned out that the code of Kyber keeps the secret key vulnerable in memory for a much shorter period of time than Frodo, resulting in a tighter window in which the Rowhammer attack had to be completed. This gives a different challenge for the Rowhammer attack. Additionally, the Rowhammer manipulation in FrodoFlips resulted in a much lower decryption failure rate, and required significantly more failing ciphertexts to obtain the key. This Kyber attack in this work requires $\sim 40,000$ ciphertexts, which can be obtained through only 4MM encryption calls. This eliminated the need of any supercomputing allowing us to instead generate all the necessary ciphertexts on a personal computer in a considerably shorter amount of time. The easier Rowhammer target and the lower requirement for failing ciphertexts enabled us to recover the full long-term secret key for Kyber.

Our attack on the BIKE KEM, on the other hand, uses a completely different approach based on flipping bits in instruction memory to disable the entropy in key generation. This does not have a parallel in the prior work on attacking PQC schemes.

Attacks on Digital Signatures. Our attack on the randomized Dilithium scheme uses an instruction flip attack

Cryptography Scheme	Purpose	Attack Summary
Kyber	KEM	Full secret key recovery using a **decryption failure attack** which was made possible through Rowhammer induced poisoning of the keypair generation process.
Dilithium-Deterministic	DSA	**Fault injection attack** leading to the nonce-reuse scenario, which enables recovery of Dilithium signing secret key.
Dilithium-Randomized	DSA	**Instruction flip attack** leading to a deterministic key generation process, which enables recovery of Dilithium signing secret key.
BIKE	KEM	**Instruction flip attack** leading to a deterministic key generation process

TABLE 1: Summary of Attacks

similar to our attack on BIKE, and for which there is no similar prior work.

Our attack on the deterministic Dilithium signature scheme is most similar to the SignatureCorrection paper of Islam et al [6]. However, our attacks are different in several fundamental ways. Our attack targets the deterministic variant of Dilithium where the Sign function does not use external randomness. The SignatureCorrection work, on the other hand, is able to attack even the randomized version of the Dilithium scheme. However, the SignatureCorrection paper only achieves leakage of partial key information, significanlty lowering the resulting security of the signature scheme (below the levels required by NIST standards) but not enabling an actual key recovery. Our attack, on the other hand, enables a full key recovery completely breaking the security of the deterministic Dilithium signature. Extending these techniques to get a full key recovery attack against the randomized variant of Dilithium is an interesting open question for future work. According to the Dilithium documentation, deterministic Dilithium is the default option except in scenarios where an adversary can mount side-channel attacks [8]. Our result suggests that the deterministic Dilithium should not be used in any setting where RowHammer is a concern (e.g., cloud deployment with shared machines).

1.4. Responsible Disclosure & Artifact Availability

We notified the authors of Kyber, BIKE, and Dilithium, providing them a preliminary copy of this paper. Source code and data are provided at the following anonymized github repository https://github.com/pqrowhammer/pqhammer

2. Background

2.1. Post-Quantum Cryptography

Post-quantum cryptography refers to cryptographic protocols that are believed to be resistant to attacks by quantum algorithms. Due to recent advances in quantum computing, there is now urgency to develop and adopt such protocols to maintain security of communications and data once such computers are available. Towards addressing this challenge, in 2016, the U.S. National Institute of Standards and Technology (NIST) announced the beginning of its Post-Quantum Cryptography (PQC) standardization process [9] aimed to standardize post-quantum public-key cryptographic algorithms including key encapsulation mechanisms (KEMs) and digital signatures. Following three rounds of review, evaluation, and cryptanalysis NIST has selected one KEM – CRYSTALS-Kyber – for standardization. They additionally designated four more KEMS: BIKE, Classic McEliece, HQC, and SIKE for further evaluation in a 4th round. NIST additionally chose three digital signature schemes: CRYSTALS-Dilithium, Falcon, and SPHINCS+ for standardization. In this paper, we study the security of three of these schemes against Rowhammer attacks.

Kyber. CRYSTALS-Kyber [10] is a lattice-based post-quantum key encapsulation mechanism (KEM) which has been chosen for standardization the third round of NIST's post-quantum cryptography standardization process [11].

BIKE. The Bit Flipping Key Encapsulation (BIKE) [12] scheme is a code-based KEM QC-MDPC (Quasi-Cyclic Moderate Density Parity-Check) codes, and is an alternate KEM in NIST's fourth round of the post-quantum cryptography standardization process [11].

Dilithium. CRYSTALS-Dilithium [13] is a lattice-based post-quantum digital signature scheme which has been chosen for standardization the third round of NIST's post-quantum cryptography standardization process [11].

2.2. The Rowhammer Bug

The Rowhammer bug is a physical vulnerability in DRAM, where repeated activations to a row cause bit flips in nearby rows [5]. The rows that are repeatedly accessed are referred to as "aggressor rows" and the rows in which bit flips are induced are referred to as "victim rows".

Double-Sided Hammering. Double-sided hammering is a Rowhammer technique where a victim row is sandwiched between two aggressor rows. The bit flips in the victim row depend on the data in the aggressor rows. For example, 1s in the victim rows are more likely to flip when the aggressor rows' values are set to 0.

Rowhammer on DDR4 Memory. As shown by Frigo et al. [14], DDR4 consumer platforms rely on in-DRAM implementations of Target Row Refresh (TRR) to mitigate Rowhammer induced bit flips. However, this mechanism can only track up to a certain number of rows, and it fails to mitigate Rowhammer when there are many aggressors.

Therefore, on DDR4, many-sided hammering is used where multiple aggressors in the same bank are hammered. Further, Kang et al. [15] showed that mutli-bank hammering, where multiple aggressors in separate banks are hammered, results in a higher number of bit flips. In this work, we employ multi-bank hammering to flip bits.

Rowhammer History. Despite continuous efforts by DRAM vendors, the Rowhammer bug still plagues the industry as researchers find new access patterns that cause bit flips in DDR4 and DDR5 memory [14]–[16]. Since Kim et. al [5] first revealed the vulnerability, numerous research has shown how it can be exploited either natively or through the browser to escalate privilege or escape sandboxes [15], [17]–[19], break RSA signature validations [20], induce faults over the network [21], and recover TLS signing keys [22]. Additionally, recent work has shown how Rowhammer can be used to mount fault injection attacks on FrodoKEM [7] and Dilithium [6].

2.3. Rowhammer Exploitation Challenges

Rowhammer Timing. Flipping bits in memory requires a sufficiently large time window to allow for enough row activations. To ensure we have such a time window, we degrade the performance of the victim process by flushing its instructions from the cache hierarchy.

Memory Profiling. We need to find a memory page that has bit flips in our desired locations. For that, the first step in our attack is to template memory by hammering in our address space until we find a page with bit flips in the correct direction and page offset.

Memory Massaging. After we have found a memory page with the required flips, we need to ensure that the victim's targeted data is allocated in that memory page. To overcome this challenge we use the technique presented in [23]. The linux kernel maintains a first-in-last-out page frame cache (PFC) for each core. Therefore, if an attacker unmaps the vulnerable page followed n dummy pages, where n is the number of pages allocated for the victim before the targeted page, and then immediately triggers the victim process, the targeted page of the victim is allocated the vulnerable memory page.

2.4. Notation

We use R to denote the ring $(\mathbb{Z}[x]/x^n+1)$, R_q to denote the ring $(\mathbb{Z}_q[x]/x^n+1)$, R_q^k to denote the space of length-k vectors where all the elements are in R_q, and $R_q^{k\times l}$ to denote the space of $k \times l$ matrices where all the elements are in R_q. We use lower-case bold letters to represent vectors and upper-case bold letters to denote matrices. $\mathbf{v}^T$ (or $\mathbf{A}^T$) represents the transpose of a vector $\mathbf{v} \in R_q^k$ (or a matrix $\mathbf{A} \in R_q^{k\times l}$).

We use $\mathbf{v}[i]$ to denote the i-th polynomial of a vector $\mathbf{v}$. We use $p[i]$ to denote the i-th coefficient of a polynomial p and $\mathbf{v}[i][j]$ to denote the j-th coefficient of $\mathbf{v}[i]$, the i-th polynomial in vector $\mathbf{v}$.

We use $\lceil x \rfloor$ to denote rounding x to the closest integer. If the rounding ties, we round up.

We use S_η to denote the uniform random distribution where a number is sampled from the range $[-\eta, \eta]$. If a vector of polynomials ($\mathbf{s_1} \in R_q^k$) is sampled from S_η^k, every coefficient of every polynomial in $\mathbf{s_1}$ is sampled from S_η.

For an odd integer q, we define $r' = r \bmod^{\pm} q$ to be the unique element in the range $-\frac{q-1}{2} \leq r' \leq \frac{q-1}{2}$ such that $r' = r \mod q$.

3. Threat Model and Hardware Setup

We assume a standard Rowhammer threat model, of an unprivileged user being able to execute code on a machine co-located with the victim. We also assume that the machine's memory is susceptible to Rowhammer-induced flips. In particular, all the techniques used in our attacks, such as performance degradation, cache flushing, memory profiling and memory massaging require no elevated privileges.

Next, Table 2 summerizes our hardware setup for each of the attacks described in this paper. Unless specified otherwise, we left all software countermeasures, such as ASLR and DRAM refresh timing in their default settings.

Scheme	CPU	Memory
Kyber	i7-6700, i7-8700K, i7-10700K	M378A1K43BB1-CPB
Dilithium	i7-8700K	M378A1K43BB2-CRC
BIKE	i7-6700, i7-8700K, i7-10700K	M378A1K43BB2-CRC

TABLE 2: Summary of attack setups (all memory part numbers are of Samsung DIMMs).

4. Decryption Failure Attack on Rowhammer-Poisoned Kyber

We give an overview of our rowhammer-assisted decryption-failure attack on KyberKEM. The scheme has a known characteristic that an honestly generated ciphertext with an honestly generated key pair has a non-zero probability of failing the decapsulation/decryption process. In that case, the two parties using the KEM fail to establish the same session key. It's been known that such an event leaks significant information about the decryption key to the encapsulator. Roughly speaking, this is because the decryption key may be viewed as a vector of small integers and decryption failure/success depends on the magnitude of the dot product between the decryption key and another vector of small integers generated during encapsulation. Therefore, decryption failure indicates that the vector known to the encapsulator is an approximation of the decryption key. However, decryption failures are extremely unlikely with probability upperbounded by $2^{-128}, 2^{-192}$, and 2^{-256}, for each respective security level. Moreover, KyberKEM is IND-CCA. An adversary seeking to exploit decryption failures through maliciously and carefully crafted ciphertexts is still very unlikely to succeed due to a mechanism called the Fujisaki-Okamoto transform and re-encryption check. For a detailed analysis of such mechanism, see [24].

Rowhammer-Induced Decryption Failure. Given the extremely low probability of obtaining a decryption failure, our Rowhammer attack aims to make decryption failure a much more likely event (i.e failure-boosting). This is achieved by exploiting the bit representation of integers, specifically the integers in the secret key. Under normal circumstances, these secret integers are small in magnitude (between -2 and 2). By flipping a single bit, a secret integer can become as large as 256. This makes the aforementioned dot product larger than usual and more likely to lead to decryption failure. We note that care must be taken in calculating which bits to flip. Otherwise, the decryption failure rate may be too high and easily noticeable or even render the key pair inoperable. Assuming that the Rowhammer attack has succeeded in flipping the desired bits, our attack proceeds by sending the victim many ciphertexts to be decrypted and subsequently observing which ciphertexts fail decryption. Each decryption failure gives us a hint or an approximation of the decryption key (cf. Equation 6). By collecting many such hints, we successfully cryptanalyze the victim retrieving the decryption key. The cryptanalysis computation involves taking the average of the many approximations of the decryption key given by decryption failures (cf. Equation 7).

Rowhammer Challenges. To mount our decryption failure attack, we introduce additional noise into the secret key during the key generation process through Rowhammer induced bit flips. Accomplishing these bit flips required overcoming several significant challenges. First, we must flip the bits within a constrained time window. Second, we must find a memory page that has the required bit flips at a suitable page offset (which depends on the page offset of the targeted variable). Finally, we must ensure that the targeted variable is allocated on the chosen vulnerable page.

4.1. Preliminaries on Kyber KEM

We provide a simplified description of KyberPKE and KyberKEM. KyberPKE is an IND-CPA PKE scheme based on Module-Learning with Error. In turn, KyberKEM is the result of applying a Fujisaki-Okamoto transform to KyberPKE. Our description of Kyber is given in Figures 1 and 2. Explanations for the $Compress$ and $Decompress$ functions as well as the Number-Theoretic Transform can be found in Kyber specification [25]. For convenience, we summarize the details below. Note that we interchange each element of R with the tuple of its coefficients; that is, we interchange R with $\mathbb{Z}^n$. Moreover, we interchange R^k with $\mathbb{Z}^{nk}$.

Compress and Decompress The two functions $Compress$ and $Decompress$ are defined as follows.

$$Compress_q(x, d) = \lceil (2^d/q) \cdot x \rfloor \bmod 2^d, \qquad (1)$$

$$Decompress_q(x, d) = \lceil (q/2^d) \cdot x \rfloor \qquad (2)$$

Essentially, $Compress_q$ approximates an integer $x \in \mathbb{Z}_q$ using d bits. The $Decompress_q$ tries to invert the compression, but the result may introduce a small error. Let $x \in \mathbb{Z}_q$ and $x' = Decompress_q(Compress_q(x, d), d)$. Then, $|(x' - x) \bmod^{\pm} q| \leq \left\lceil \frac{q}{2^{d+1}} \right\rfloor$.

Moreover, $Compress_q$ and $Decompress_q$ can be used as a sort of error-correcting code. Let $m \in \{0, 1\}$ and $m' = Decompress_q(m, 1) + e \bmod q$. Then if $|e| \leq \lceil q/4 \rfloor$, we obtain that $m = Compress_q(m', 1)$.

Centered Binomial Distribution The centered binomial distribution B_η over $\mathbb{Z}$ is the distribution of the output of the following sampling procedure:

1) Sample $(a_1, \ldots, a_\eta, b_1, \ldots, b_\eta) \leftarrow \{0, 1\}^{2\eta}$
2) Output $\sum_{i=1}^{\eta}(a_i - b_i)$.

Equivalently, B_η is the binomial distribution with 2η trials and success probability $1/2$, shifted by $-\eta$.

Number-Theoretic Transform Specifically, for Kyber, $n = 256$ and $q = 3329 = 13 \cdot 2^8 + 1$, which is a prime. So $x^n + 1$ is the 512th cyclotomic polynomial, whose zeros are the 256 primitive roots of unity of order 512.

Over a sufficiently large extension of $\mathbb{Z}_q$, the polynomial $x^n + 1$ factors into 256 linear factors whose zeros are all primitive roots of unity of order 512. However, the field $\mathbb{Z}_q$ does not contain a primitive root of unity of order 512 but does contain a root of unity of order 256, e.g. $\zeta = 17$. Therefore, over $\mathbb{Z}_q$, the polynomial $x^n + 1$ factors into 128 irreducible polynomials of degree 2:

$$x^n + 1 = \prod_{i=0}^{127}(x^2 - \zeta^{2i+1}).$$

Therefore, the ring R_q is isomorphic to the product ring $\prod_{i=0}^{127} \mathbb{Z}_q[x]/\langle x^2 - \zeta^{2i+1}\rangle$,

$$R_q \cong \mathbb{Z}_q/\langle x^2 - \zeta\rangle \times \mathbb{Z}_q/\langle x^2 - \zeta^3\rangle \cdots \times \mathbb{Z}_q/\langle x^2 - \zeta^{255}\rangle.$$

The isomorphism is given by

$$f \mapsto (f \bmod x^2 - \zeta, \ldots, f \bmod x^2 - \zeta^{255}).$$

The image of f under the isomorphism mapping is called the number-theoretic transform (NTT) of f, denoted by $\mathrm{NTT}(f)$. The inverse function, as usual, is denoted by NTT^{-1}.

4.2. Decryption Failure Rate Analysis

In this section, we calculate the decryption failure rate when only two bits are flipped via rowhammer during key generation. As previously discussed, the Kyber KEM may generate ciphertexts that do not decrypt correctly, i.e. decryption failures. In this section, we describe the conditions under which decryption failures occur and analyze the probability that they occur given that the error vector $\mathbf{e}$ is rowhammered during KeyGen. We show that by rowhammering the vector $\mathbf{e}$ at only 2 locations, we can increase the failure rate to be noticeable. However, the failure rate is still sufficiently low for the key pair to appear operational to honest users.

In more detail, let $\mathbf{c}_u = \mathbf{u} - \mathbf{u}'$ and $c_v = v - v'$. Decryption is correct if

$$\|\mathbf{e}^T\mathbf{r} - \mathbf{s}^T\mathbf{e}_1 - \mathbf{s}^T\mathbf{c}_u + e_2 + c_v\|_\infty < \lceil q/4 \rfloor.$$

Algorithm 1 KyberPKE.KeyGen()

```
1: $\hat{\mathbf{A}} \leftarrow \mathrm{NTT}(R_q)^{k\times k}$
2: $\mathbf{s} \leftarrow (B^n_{\eta_1})^k$ , $\mathbf{e} \leftarrow (B^n_{\eta_1})^k$
3: $\hat{\mathbf{s}} = \mathrm{NTT}(\mathbf{s})$
4: $\hat{\mathbf{e}} = \mathrm{NTT}(\mathbf{e})$
5: $\hat{\mathbf{t}} = \hat{\mathbf{A}} \cdot \hat{\mathbf{s}} + \hat{\mathbf{e}}$
6: return $(pk = (\hat{\mathbf{A}}, \hat{\mathbf{t}}), sk = \hat{\mathbf{s}})$
```

Algorithm 2 KyberPKE.Enc(pk, μ)

```
1: $(\hat{\mathbf{A}}, \hat{\mathbf{t}}) = pk$
2: $\mathbf{r} \leftarrow (B^n_{\eta_1})^k$
3: $\mathbf{e}_1 \leftarrow (B^n_{\eta_2})^k$
4: $e_2 \leftarrow B^n_{\eta_2}$
5: $\hat{\mathbf{r}} = \mathrm{NTT}(\mathbf{r})$
6: $\mathbf{u} = \mathrm{NTT}^{-1}(\hat{\mathbf{A}} \cdot \hat{\mathbf{r}}) + \mathbf{e}_1 \bmod q$
7: $v = \mathrm{NTT}^{-1}(\hat{\mathbf{t}} \cdot \hat{\mathbf{r}}) + e_2 + Decompress_q(\mu, 1) \bmod q$
8: $c_1 = Compress_q(\mathbf{u}, d_u)$
9: $c_2 = Compress_q(v, d_v)$
10: return $c = (c_1, c_2)$
```

Algorithm 3 KyberPKE.Dec(sk, c)

```
1: $\hat{\mathbf{s}} = sk$
2: $(c_1, c_2) = c$
3: $\mathbf{u}' = Decompress_q(c_1, d_u)$
4: $v' = Decompress_q(c_2, d_v)$
5: $m = v - \mathrm{NTT}^{-1}(\mathbf{s}^T \cdot \mathrm{NTT}(\mathbf{u}))$
6: return $\mu = Decompress_q(m, 1)$
```

Figure 1: Simplified description of Kyber PKE

Algorithm 4 KyberKEM.KeyGen()

```
1: $(pk', sk') \leftarrow$ KyberPKE.KeyGen()
2: return $(pk = pk', sk = (pk', sk'))$
```

Algorithm 5 KyberKEM.Enc(pk)

```
1: $\mu \leftarrow \{0,1\}^\ell$
2: $(r, k) \leftarrow H(\mu)$
3: $c \leftarrow$ KyberPKE.Enc$(pk, \mu; r)$
4: return $(c, k)$
```

Algorithm 6 KyberKEM.Dec(sk, c)

```
1: $(pk', sk') = sk$
2: $\mu \leftarrow$ KyberPKE.Dec$(sk', c)$
3: $(r, k) \leftarrow H(\mu)$
4: $c' \leftarrow$ KyberKEM.Enc$(pk, \mu; r)$
5: if $c' = c$ then
6:     return $k$
7: else
8:     Decapsulation fails.
```

Figure 2: KyberKEM as Fujisaki-Okamoto transform of KyberPKE

On the other hand, if

$$\|\mathbf{e}^T\mathbf{r} - \mathbf{s}^T\mathbf{e}_1 - \mathbf{s}^T\mathbf{c}_u + e_2 + c_v\|_\infty > \lceil q/4 \rfloor,$$

then decryption failure overwhelmingly likely occurs [26]. Let $E = \mathbf{e}^T\mathbf{r} + e_2 + c_v - \mathbf{s}^T\mathbf{e}_1 - \mathbf{s}^T\mathbf{c}_u$. We shall call this the decoding error. We are interested in the event that one of the coefficients of E exceeds $q/4$ in magnitude, which will cause a decryption failure.

We interchange each element of R with the tuple of its coefficients; that is, we interchange R with $\mathbb{Z}^n$. Moreover, we interchange R^k with $\mathbb{Z}^{nk}$. We note that there is a natural correspondence between each column $\mathbf{t} \in R^k$ and the $\mathbb{Z}$-matrix representation of the linear map $\mathbf{x} \mapsto \mathbf{t}^T\mathbf{x}$ where $\mathbf{x} \in R^k$ when this map is viewed as a $\mathbb{Z}$-linear map of $\mathbb{Z}^{nk}$.

Let ψ_u and ψ_v be the distributions of the coefficients of $\mathbf{c}_u$ and c_v respectively. Under the hardness assumption of decisional Module-Learning with Error, ψ_u can be sampled as follows:

1) $u \leftarrow \mathbb{Z}_q$
2) Output $u - Decompress_q(Compress_q(u, d_u), d_u)$

And ψ_v can be sampled similarly. From the equation above for E, the distribution of the coefficients of E then is

$$(B_{\eta_1} \cdot B_{\eta_1})^{\oplus kn} + (B_{\eta_1} \cdot (B_{\eta_2} + \psi_u))^{\oplus kn} + B_{\eta_2} + \psi_v.$$

where for a distribution B, $(B)^{\oplus kn} = \underbrace{B + B + B}_{kn \text{ times}}$.

Now suppose we flip 2 bits of coefficients in $\mathbf{e}$, one at the 2^8 bit and one at the 2^6 bit, forcing these bits to be set. We calculate that when two such bits are rowhammered, the decryption failure rate is sufficiently high to allow for practical key-recovery. Nonetheless, the decryption failure rate is still sufficiently low for the generated key pair to appear operational to honest users. In the following, we assume that these two coefficients are non-negative. That is, the targeted bits are 0. This is a high-probability event. Then, we further assume that the rowhammer succeeds in flipping these bits to 1. The two affected coefficients then have following distributions respectively:

$$H^{256}_{\eta_1}(256 + x) = \begin{cases} B_{\eta_1}(0) & \text{if } x = 0 \\ 2B_{\eta_1}(x) & \text{if } x > 0 \end{cases}$$

and

$$H^{64}_{\eta_1}(64 + x) = \begin{cases} B_{\eta_1}(0) & \text{if } x = 0 \\ 2B_{\eta_1}(x) & \text{if } x > 0 \end{cases} .$$

Then, the distribution of the coefficients of E for honest users becomes

$$H^{256}_{\eta_1} \cdot B_{\eta_1} + H^{64}_{\eta_1} \cdot B_{\eta_1} + (B_{\eta_1} \cdot B_{\eta_1})^{\oplus(kn-2)} + (B_{\eta_1} \cdot (B_{\eta_2} + \psi_u))^{\oplus kn} + B_{\eta_2} + \psi_v. \quad (3)$$

Numerically calculating the probability p that the distribution in equation 3 exceeds the decoding threshold gives $p \approx 2^{-18}$. If we assume that each coefficient of the decoding error is independent, an honest user's decryption failure rate is $1-(1-p)^{256} \approx 2^{-10}$. We have experimentally validated the decryption failure rate to be slightly higher than 2^{-10}, landing just under $2^{-10.5}$ for our rowhammered keys.

Now consider an adversary engaging in failure boosting. Our failure boosting technique consists of repeatedly sampling $\mathbf{r}$ until the two coefficients of $\mathbf{r}$ that get multiplied with the rowhammered coefficients of $\mathbf{e}$ in the equation for E, have the maximum magnitude and the same sign. The sum-product of these two coefficients of $\mathbf{r}$ with the two rowhammered coefficients of $\mathbf{e}$ then has the following distribution K:

1) With probability $1/2$, output a sample from $\eta_1 \cdot H^{256}_{\eta_1} + \eta_1 \cdot H^{64}_{\eta_1}$
2) Else, output a sample from $-\eta_1 \cdot H^{256}_{\eta_1} - \eta_1 \cdot H^{64}_{\eta_1}$

One of the coefficients of decoding error E for the adversary is distributed as

$$K + (B_{\eta_1} \cdot B_{\eta_1})^{\oplus(kn-2)} + (B_{\eta_1} \cdot (B_{\eta_2} + \psi_u))^{\oplus kn} + B_{\eta_2} + \psi_v. \quad (4)$$

The coefficient with the distribution above exceeds the decoding threshold with probability $p' \approx 2^{-11}$. Let us call this coefficient E_k. Let us heuristically assume that each of the other 255 coefficients of E is independent and exceeds the decoding threshold with probability (approximately) equal to that of equation 3. The fraction of failing ciphertexts where E_k does not exceed the decoding threshold is

$$\frac{(1-p')(1-(1-p)^{255})}{1-(1-p')(1-p)^{255}}. \quad (5)$$

Numerically, this is $\approx 67\%$.

4.3. Recovering the Secret via Decryption Failures

In this section, we give an analysis explaining our key recovery step. At a high level, taking the average of the hints given by decryption failures, we recover the secret key up to a known scaling factor. The key recovery step for Kyber is more involved than prior work on Frodo [7]. Despite having foreknowledge of which bit flips were targeted, we cannot fully determine which coefficient of the error exceeds the decoding threshold. Inevitably, ciphertext data that is uncorrelated with the secret key is included during the averaging step[1]. Maximizing our information gain therefore requires estimating the failing coefficient(s) with as high of an accuracy as possible. Our implemented estimator succeeds with $\approx 75\%$ accuracy, which was sufficient for key recovery.

For our analysis, we heuristically treat each distribution B_{η_i} as a normal distribution $N(0, \sigma_i^2)$ for some $\sigma_i \in \mathbb{R}^+$ and follow the analysis in [27] with some modifications. First we assume that rowhammer is successful, i.e. one coefficient of $\mathbf{e}$ is at least 64 and one is at least 256. Our following analysis is conditioned on this event.

1. This assumes that the coefficients in failing ciphertexts that do not exceed the decoding threshold are uncorrelated with the secret.

For any $a \in R$, we write $\mathrm{Mat}(a)$ for the $\mathbb{Z}$-matrix representation of a. If $\mathbf{a} = (a_1, a_2, \dots) \in R^*$, we define $\mathrm{Mat}(\mathbf{a})$ to be the block matrix $[\mathrm{Mat}(a_1)|\,\mathrm{Mat}(a_2)|\dots]$. Moreover, we define $\vec{\mathbf{a}}$ the column tuple obtained by concatenating (vertically stacking) the columns $\vec{a_1}, \vec{a_2}, \dots$.

For simplicity, assume that decryption failure has occurred in the first coordinate and $|E_0| > t$ where $t = \lceil q/4 \rfloor$. Let e' be the first row of $\mathrm{Mat}(\mathbf{e})$ with the two coordinates corresponding to the two rowhammered coefficients erased. Let h_1 and h_2 be these two coordinates. Let f_1 and f_2 be the two coefficients of $\mathbf{r}$ that get multiplied with h_1 and h_2 respectively in the computation of E_0. Let r' be the column obtained by erasing f_1 and f_2 from $\vec{\mathbf{r}}$. Moreover, let s' be the first row of $\mathrm{Mat}(\mathbf{s})$. Then,

$$E_0 = e' \cdot r' + f_1 \cdot h_1 + f_2 \cdot h_2 + s' \cdot (\vec{\mathbf{e_1}} + \vec{\mathbf{c}}_u) + e_2^{(0)} + c_v^{(0)}.$$

Let $S = (e', s')^T$, the column obtained by concatenating e' and s', and $W = \begin{bmatrix} r' \\ \vec{\mathbf{e_1}} + \vec{\mathbf{c}}_u \end{bmatrix}$ the column obtained concatenating r' and $\vec{\mathbf{e_1}} + \vec{\mathbf{c}}_u$. Then,

$$E_0 = \langle S, W \rangle + f_1 \cdot h_1 + f_2 \cdot h_2 + e_2^{(0)} + c_v^{(0)}.$$

It suffices to consider the case $E_0 > t$. Otherwise, if $E_0 < -t$, consider $-E_0$ instead. According to the failure boosting technique described above, $f_1 = f_2 = \pm\eta_1$. If $f_1 = f_2 = -\eta_1$, the probability that $E_0 > t$ is extremely low. So we may assume that $f_1 = f_2 = \eta_1$. We rewrite $E_0 > t$ as

$$\langle S, W \rangle > t' = t - f_1 \cdot h_1 - f_1 \cdot h_2 - e_2^{(0)} - c_v^{(0)}. \quad (6)$$

Note that the adversary has knowledge of each term for computing t' except h_1 and h_2. However, it is trivial to guess h_1 and h_2 as the set of possible values is very small. So we further assume that t' is known.

We heuristically treat S as a Gaussian $\mathcal{N}(0, \sigma^2)$. Due to the concentration of the norm of such a high-dimensional Gaussian, we assume that the norm of S is $\ell = \sigma\sqrt{N}$ where N is the dimension. For the same reason, we assume $|e'| = \sigma\sqrt{N_1}$ and $|s'| = \sigma\sqrt{N_2}$.

We also heuristically treat W as a Gaussian $\mathcal{N}(0, \Sigma)$ where

$$\Sigma = \begin{bmatrix} \tau_1^2 I & 0 \\ 0 & \tau_2^2 I \end{bmatrix}.$$

We consider the decomposition of W as the sum of its projections onto the subspace spanned by S and the subspace orthogonal to S. Let Π_S be the matrix of projection onto the former and $\Pi_\perp$ be the matrix of projection onto the latter. Then,

$$\begin{aligned} W &= \Pi_S \cdot W + \Pi_\perp \cdot W \\ &= \alpha \frac{S}{\ell} + \Pi_\perp \cdot W, \end{aligned}$$

where

$$\alpha \sim \mathcal{N}(0, \frac{\tau_1^2 \sigma^2 N_1 + \tau_2^2 \sigma^2 N_2}{\sigma^2 N}) = \mathcal{N}(0, \frac{\tau_1^2 N_1 + \tau_2^2 N_2}{N}).$$

```
1 gen_a(a, publicseed);
2 for(i=0;i<KYBER_K;i++)
3    poly_getnoise_eta1(&skpv.vec[i], noiseseed,
      nonce++);
4 for(i=0;i<KYBER_K;i++)
5    poly_getnoise_eta1(&e.vec[i], noiseseed, nonce
      ++);
6 polyvec_ntt(&skpv); // window for inducing flips
7 polyvec_ntt(&e);
```

Listing 1: Snippet of Kyber key generation code showing the initialization of $\mathbf{e}$ and conversion to NTT

Note that $\langle S, W\rangle > t'$ if and only if $\alpha > t'/\ell$. Next we have

$$\Pi_\perp \cdot W \sim \mathcal{N}(0, \Pi_\perp \Sigma \Pi_\perp^T).$$

Let τ be a number greater than or equal to both τ_1 and τ_2, e.g. $\tau = \max(\tau_1, \tau_2)$. Then $\Pi_\perp \Sigma \Pi_\perp^T \leq \tau^2 I$. Then,

$$W = \frac{\alpha'}{\ell} S + T$$

where α' has the distribution of α conditioned on $\alpha \geq t'/\ell$ and T has covariance $\leq \tau^2 I$. Taking expectation gives

$$E[W] = \frac{E[\alpha']}{\ell} S + E[T] = \frac{E[\alpha]}{\ell} S. \tag{7}$$

4.4. Rowhammer Induced Poisoning

Inducing bit flips on the reference Kyber implementation through rowhammer was met with a few challenges that are typical with Rowhammer attacks. Namely, flipping bits within a necessary time window, finding memory addresses that correspond to consecutive rows in the same bank, finding a memory page with bit flips in the necessary page offset for the attack, and forcing the victim to use the chosen vulnerable memory page. In this section, we describe how we addressed these challenges to execute our attack successfully.

Timing Constraints. To Rowhammer the relevant coefficients of $\mathbf{e}$, there is a narrow window of opportunity. The arrays for these values are initialized in memory as contiguous blocks of 0s. Injecting bit flips before this initialization therefore would have no actual effect. Kyber makes use of the number theoretic transform as a low-memory performance optimization for polynomial multiplication. The consequence with regard to bit flips is that generating a precise amount of noise in the euclidean domain would require a considerable number of precise bit flips in the NTT domain. The more bits our Rowhammer script aims to flip, the higher likelihood of incidentally flipping other bits, which requires a masking technique to prevent these unwanted flips. The combination of these two memory operations sets the time constraint for any feasible rowhammer attack. In short, the total time to add noise into the secret key is the time between matrix generation and conversion to NTT.

Static timing analysis of these functions determines that the highest number of clock cycles during this process occurs during the `polyvec_ntt` function, which in turn calls `poly_ntt`. For the Kyber-1024 process, the `poly_ntt` function is called 8 times, so it is possible to focus on the last portion of the error term (e) during the other seven runs. Each iteration of `poly_ntt` uses about 35,000 cycles. For modern machines, this process at the highest end is approximately 1400 times less than the amount of time necessary to complete a full DRAM refresh cycle.

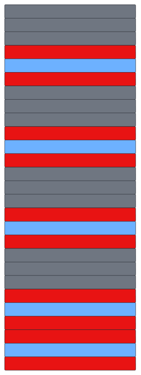

Figure 3: 10-sided hammering example. There are 5 aggressor pairs (red) where each pair is sandwiching a victim row (blue).

Performance Degradation. To address the challenge of flipping bits in the $\mathbf{e}$ matrix between matrix generation and conversion to NTT, it in necessary to slow the execution of the `polyvec_ntt` function, specifically as it executes on `skpv` (line 6 in Listing 1). To achieve this goal, we use performance degradation techniques from Hyperdegrade [28] where frequently accessed instructions are flushed from the cache hierarchy by attacker processes and pipeline stalls are induced in the physical core running the victim.

The `polyvec_ntt` function iteratively calls a multiplication followed by a montgomery reduction, enabling two potential function calls that can be evicted from instruction cache to create a cache flush. We determined from this analysis that attacking Kyber-1024 creates the longest possible time duration because it increases the total number of these function calls. Through static analysis we found that the most executed instructions are inside the `montgomery_reduce` function called by `ntt` function in a loop. Therefore, we chose one cacheline inside the `montgomery_reduce` function. And two cachelines in the `ntt` function.

We run many instances of processes that flush instructions of the victim process from the cache hierarchy and processes that perform a spin loop. The processes are distributed between all the processors (except for the physical core hammering the aggressors). We pin the processes flushing the most frequently accessed cachelines to the victim sibling core, which has the added benefit of inducing machine clears and thus further degrading the victim

Contiguous Memory. To obtain bit flips on DDR4 we need 10 aggressors, and each pair of aggressors must sandwich a victim row between them as shown in Figure 3. To find aggressor pairs (two aggressor rows with a victim in between them) we must first find physically contiguous memory. We observed that Transparent Huge Pages (THPs) were not massageable for the Kyber attack, which targets a variable on the stack of the victim process. Therefore, we used SPOILER [29] to detect contiguous memory within an allocated memory region. SPOILER uses false load dependencies due to aliasing in the lower 20 bits of physical addresses to detect contiguous memory. However, we observed that SPOILER cannot differentiate between a single 2MB physically contiguous memory block and a 2MB memory block that contains two separate 1MB physically contiguous blocks where both start at the same 2MB alignment due to similar aliasing effects. Therefore, we use SPOILER to detect multiple sets of contiguous 256 pages. SPOILER uses a single arbitrary load address (referred to as x in [29]) to detect the aliasing effect. Setting this address to a 2MB aligned address (obtained through THPs and only used as the load address of SPOILER), we can ensure that the contiguous memory we detect is 2MB aligned.

Finding Aggressors. After obtaining contiguous memory, we need to find memory addresses that reside in the same bank. The bank address bits, which we found using DRAMA [30], are all within the lower 20 bits of the physical address. Since the memory we obtained is 2MB physically aligned, we know all the bits needed to find the bank number of any memory address. Using this fact, we create a list of aggressor pairs for each bank and set the aggressors and corresponding victims to a striped pattern (1-0-1).

Multi-Bank Hammering. Kang et. al [15] showed that bank-level parallelism can be used to amplify Rowhammer by hammerring aggressors in multiple banks. The hammering pattern in [15] accesses the first aggressors in all the hammered banks followed by the second aggressor, etc.

Memory Profiling. We now perform multi-bank hammerring on two banks using 10 aggressors in each bank. Notably, we use the `clflushopt` instruction to flush the aggressors' data from the cache hierarchy.

Memory Massaging. The attacker must now force the victim to allocate $\mathbf{e}$ in the chosen page. As described in [23] we perform "Frame Feng Shui" to massage the chosen page to the correct location in the victim address space. We unmap a number of dummy pages before triggering the process, observe on which dummy page the $\mathbf{e}$ has been allocated and replace that dummy page with the chosen victim page.

4.5. Experimental Setup and Results

Setup. We ran our experiment on a desktop containing an Intel i7-8700K (Coffee Lake) CPU and a single Samsung DDR4 8GiB DIMM (part number M378A1K43BB1-CPB). Our machine was running Ubuntu 23.04. For debugging visibility only, processes ran using sudo. We used the Kyber 1024 reference code [31], compiled using the reference makefile, as our victim. Since we target Kyber's e matrix which is allocated on the stack, its location inside a memory page is randomized by the operating system's Address Space Layout Randomization (ASLR). Thus, below we test for two configurations, namely with and without ASLR.

Results. We ran our attack 100 times with ASLR both on and off. With ASLR off, our chosen page was successfully massaged 91 times and the correct bits (and only the correct bits) of the targeted variable were flipped 8 times via Rowhammer. With ASLR on, our chosen page was successfully massaged 44 times and out of those the correct bits of the targeted variable were flipped 3 times. These results show the effect ASLR has on shifting the page offset of the stack variable and reducing massaging success rate.

On average, the profiling step of our attack takes 49 seconds and the massaging and hammering takes 93 seconds due to the performance degradation.

Attack Perceptibility. To test the perceptibility of our performance degradation, we ran the Geekbench 6 benchmark on our machine with and without performance degradation. Without any degradation, our machine scored 1674 points for single-core performance and 5600 points for multi-core performance. With performance degradation running, these become 1543 points for single-core and 1648 points for multi-core. That is, we observed an 8% reduction in single-core performance, and a 71% reduction in multi-core score. In all cases, the machine was snappy and reactive, without any noticeable lag in user experience.

Generating Failing Ciphertexts. A consequence of the selected number of bit flips is an increase in decryption failure rates. Our noise level increases Keyber's honestly generated failure rate to $\sim \frac{1}{10000}$. The Kyber construction allows for key recovery from this failure rate with only approximately 40k failing ciphertexts, taking about 2 hours to generate on a modern PC. These ciphertexts were saved to files based and processed using the method described in Section 4.3 and [7].

Key Extraction. To extract the secret key, we estimate the failing coefficient for each ciphertext and rotate their vector representation correspondingly when averaging. We use the James-Stein estimator to obtain the final scaling factor and recover the key. Each recovery iteration averages 4.5 minutes on a modern server (Intel Xeon Gold 5420+). For ASLR-disabled victims with bit flips in known locations a recovery attempt takes a single iteration, allowing us to recover the keys from all 8 successful attack runs described above. For ASLR-enabled case with a known bit separation, a recovery will occur during one of 1024 attempts, averaging 512 attempts. From the 3 successful attack attempts above, 2 of our keys completed in 33 and 940 iterations respectively. The 3rd key did not have known bit separation for the two errors, and recovery is upper-bound to 1024×1024 iterations. With each iteration lasting about 4.5 minutes, we estimate key recovery time to be about 60 core-months on average for our third attack attempt. Thus, while our key extraction method is parallelizable across different iterations, we leave the task of creating a more efficient key recovery algorithm with unknown bit separation to future work.

5. Rowhammer Attack on BIKE-KEM

This section describes an instruction memory flip attack that can be launched against the reference implementation of the BIKE algorithm.

Instruction Flip Attacks. The von Neumann computer architecture–on which most modern computers are based–uses a single shared memory for both program and data memory. This allows Rowhammer attacks to flip instruction bits (in addition to data), causing victim programs to operate differently than their intended purpose. As opposed to data memory which can potentially induce unintended bit flips that don't affect the program, instruction hammering must be precise in order to achieve the desired effect. Single bit flips in the operation code or operands on the instruction word can cause significant deviation from the intended effect. However, instruction hammering has the advantage that intentional selections of bits in instruction memory can allow attackers to manipulate which registers are used in computation, change memory addresses for reads or writes, or change the kind of computation that occurs. This has been shown in prior literature [15], [17] to intentionally manipulate control flow.

Memory Deduplication. To conserve memory usage, operating systems point the virtual pages of processes reading the same data to the same physical pages in memory, thus allowing the data to be stored in memory only once. Previous work explored how memory deduplication can leak information about processes through side-channels. Yarom et. al [32] shows how Flush+Reload can be used to leak information about process control flow. Gruss et. al [17] also show how Flush+Reload can be used to find if a victim code page is allocated at a certain physical address. In our instruction flip attack, memory deduplication ensures that the victim process uses the corrupted binary as its code.

Instruction Flip Attacks on BIKE. Using the same process as Kyber, the scheme and implementation were inspected for potential assumptions that could be violated through bit flip attacks, this time focusing on instruction memory. We determined that instruction flip attacks could be levied against an assumption in the scheme that the seed passed to the pseudorandom function is the random number generated by the hardware. Note that this is a different attack than intentionally manipulating the pRNG or RNG on hardware because the randomness from the hardware function can be validated for correctness and the attack can still succeed. Instead, we manipulate the location where the seed is stored prior to iterative calls to a pseudorandom function. In the following sections we describe an end-to-end attack on the pseudorandom number used as entropy during key generation produce a deterministic key.

BIKE Key Generation. The BIKE scheme is based on the McEliece scheme by instantiating "Quasi-Cyclic Moderate Density Parity Check codes". The private key is generated through a pseudorandom mechanism to a set a small number of bits in a zero-indexed vector to produce a sparse uniform distribution. This process is conducted twice to produce h_0 and h_1, and the public key is constructed as $H = h_1 h_0^{-1}$ as an inversion and multiplication in the finite field. To produce the two sparse vectors, the specification and reference code leverage a pseudorandom function based on the SHAKE256 algorithm. The reference implementation of BIKE initializes the pseudorandom function state by way of a function call to the system's random function. From this point forward, *the state of the pseudorandom function is entirely dependent on the initialized seed.*

Attacking the pseudorandom seed. Using instruction memory flips, we were able to launch an end-to-end key recovery attack with the aim of intentionally selecting the seed value. Arbitrarily setting the seed value enables the attacker to fully replicate the SHAKE256 function and obtain the secret key, validating its correctness with the public key of the victim. We describe our experiment in Subsection 5.

Rowhammer. Similar to [17], we perform a Rowhammer attack on the binary of the BIKE key pair generation function. Listing 2 and Listing 3 show the assembly dump and the C code of the relevant sections of the BIKE-KEM key generation process.

We use the reference implementation test program as our victim. Our attack leads to a deterministic key generation. We target a bit flip in the operands of the `mov` instruction that sets up the arguments for the `get_seeds` functions (line 3 in Listing 2). We found that flipping 89 to either 88 or flipping df to either de or db results in a seed of 0 and a deterministic keypair without any errors given by the program.

```
1 0000000000001760 <crypto_kem_keypair>:
2   ...
3   1848:  48 89 df        mov    %rbx,%rdi
4   184b:  e8 00 e8 00 00  callq  10050 <get_seeds>
5   ...
```

Listing 2: Assembly dump snippet of BIKE key generation

```
int crypto_kem_keypair(OUT unsigned char *pk, OUT
    unsigned char *sk){
    ...
    get_seeds(&seeds);
    GUARD(generate_secret_key(&h0, &h1,l_sk.wlist
    [0].val, l_sk.wlist[1].val,&seeds.seed[0]));
    ...
}
```

Listing 3: BIKE key generation Code

Experimental Setup. We used a setup similar to our Kyber attack (see Section 4) with a desktop running Ubuntu 22.04 and containing a single Samsung DDR4 8GiB DIMM (part number M378A1K43BB2-CRC). We used the test program of the reference implementation of BIKE as our victim [33] Our attack process ran with normal user privileges.

Contiguous Memory. We observe that THPs are massagable for this attack unlike the Kyber attack (see section 4) because the allocation is done through an `mmap` call (which was not the case for the Kyber attack). THPs are available through madvise and allow us to allocate multiple contiguous and physically aligned 2MB memory blocks. Using these memory blocks we find aggressor pairs in all the banks using the bank address bits as we did in the Kyber attack (see Section 4)

Memory Profiling. We perform multi-bank hammering on two banks in search of bit flips at the page offsets where our targeted bytes would reside.

OS Page Cache. The page cache is a software collection of disk-backed backed pages such as program binaries and shared libraries that the operating system may keep in memory to service processes that may read them [34]. When multiple processes are reading (and only reading) the same file, their respective virtual address spaces will point to the same physical page frames containing the file. We exploit the fact that there is only one copy of the binary in the page cache, meaning that it is shared between the victim and attacker. As a result, any surreptitious modifications the attacker makes to the binary (e.g. via Rowhammer) are reflected in the victim's binary.

Memory Massaging. After finding a suitable victim page we call madvise with the MAP_PAGEOUT flag on the base page of the THP that the victim page belongs to, which breaks down the THP into normal 4KB pages, as explained in [15], thereby making it usable for massaging. Next, we unmap the page followed by a single dummy page, as the targeted flips are in the second page of the binary and the Linux PFC follows a first-in-last-out structure. We then map the binary into the attacker's address space with read-only privileges and the MAP_POPULATE and MAP_SHARED flag set. This ensures that the targeted page of the binary is allocated on the chosen victim page. Since we map the binary as read-only, this brings it into the page cache and ensures that any other process reading the same binary file will be pointed to the same page in memory as our attack process. However, if the binary was already in the page cache (i.e. in memory) the Linux kernel will not bring it in again and instead point our virtual addresses to that physical memory. Therefore, we run a memory exhaustion process before we run our attack to ensure that the binary has been evicted from memory.

Hammering. Now that the targeted binary code is in the victim page we hammer the aggressors until the desired bit flip occurs (or we reach a set number of tries to prevent running infinitely), which we are able to check as the binary file is mapped into our address space with read-only privileges. After the binary flips we trigger the victim program which will use the modified binary in the page cache as its instructions, as shown in Figure 4. Finally, we note that no performance degradation was required for this attack.

Experimental Results. We found a total of 198 unique bit flips before finding a bit flip at the correct page offset. After the binary was massaged onto the target page the hammering pattern was performed five times until the bit flip was replicated on the target binary page. We then triggered the victim process which outputted a deterministic keypair.

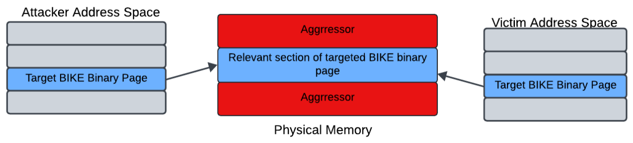

Figure 4: Memory layout after the targeted binary page of BIKE has been massaged. Only the relevant section of the page is in the victim row as Intel's Coffee Lake DDR4 DRAM address mapping splits pages between two banks. Both the attacker and the victim are pointing to the same physical page as both processes are only reading.

6. Rowhammer Attack on Dilithium

6.1. Crystals-Dilithium

Dilithium. CRYSTALS-Dilithium (ML-DSA) is a post-quantum digital signature scheme selected by the National Institute of Standards and Technology (NIST) [8]. Dilithium is based on the "Fiat-Shamir with Aborts" technique [8], where it samples a secret nonce for every signature generation. Leaking the secret nonce directly reveals the secret key and thus breaks Dilithium [35]. The scheme is deterministic when it samples the secret nonce deterministically, and randomized when it samples the secret nonce randomly. This implies that when signing the same message multiple times, deterministic Dilithium generates the same nonce, and randomized Dilithium generates a new nonce every time.

In Algorithm 7, we present a simplified version of the deterministic Dilithium signing algorithm, highlighting the components relevant to our Rowhammer attack. Dilithium maintains a secret key $\mathbf{s_1} \in R_q^l$. When signing a message M, it first expands a public seed ρ into a public matrix $\mathbf{A}$ (Line 1). Then, it enters a while loop, inside which it generates a new secret nonce $\mathbf{y}$ deterministically depending on the M and loop counter (Line 3), a challenge sparse polynomial c depending on $\mathbf{A}$, $\mathbf{y}$, and M (Line 4), and computes the final signature as $(\mathbf{z} := \mathbf{y} + c\mathbf{s}_1, c)$ (Line 5). It checks whether $\mathbf{z}$ leaks information about the secret key and rejects $\mathbf{z}$ if it does. The while loop terminates when $(\mathbf{z}, c)$ passes the safety check and is secure to output.

Variable $\mathbf{s_1}$ and c have low Hamming weight, whereas $\mathbf{z}$ and $\mathbf{y}$ have high Hamming weight.

6.2. Attacking Deterministic Dilithium

In this section, we demonstrate that strategically injecting faults at precise positions by massaging the memory carefully enables a successful recovery of the entire Dilithium secret key $\mathbf{s_1}$. We assume deterministic Dilithium.

Attack Core Idea. First, we present the core idea of our high-level attack methodology. For a given message M, assume we obtain both a correct signature $(\mathbf{z}, c)$ and a faulty signature $(\mathbf{z}', c')$ that are generated using the same secret nonce $\mathbf{y}$:

$$\mathbf{z} := \mathbf{y} + c\mathbf{s}_1 \tag{8}$$
$$\mathbf{z}' := \mathbf{y} + c'\mathbf{s}_1 \tag{9}$$

An attacker can easily recover the secret key $\mathbf{s}_1$ as:[2]

$$\mathbf{s}_1 = \frac{\mathbf{z} - \mathbf{z}'}{c - c'} \tag{10}$$

2. $c - c'$ is invertible with very high probability [36]

Algorithm 7 A simplified version of the Dilithium signing algorithm. The algorithm contains a while loop that samples a secret nonce $\mathbf{y}$ deterministically based on M, the message being signed, and generates a signature $(\mathbf{z}, c)$. It outputs $(\mathbf{z}, c)$ if it leaks no information about the secret key.

```
Require: Secret key $\mathbf{s}_1$, and message $M$
1: $\mathbf{A} \in R_q^{k \times l}$ := ExpandA($\rho$)             ▷ Expand a public seed $\rho$ to a public matrix $\mathbf{A}$.
2: while $\mathbf{z} = \perp$ do
3:     $\mathbf{y} \in R_q^l$ := SampleY($M, \mathbf{s}_1$)          ▷ Sample a secret nonce $\mathbf{y}$ based on $M$.
4:     $c \in R_q$ := SampleC($\mathbf{A}, \mathbf{y}, M$)           ▷ Sample a polynomial $c$ based on $\mathbf{A}$, $\mathbf{y}$, and $M$.
5:     $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$
6:     if $\mathbf{z}$ leaks information about the secret key $\mathbf{s}_1$ then   ▷ Safety check.
7:         $\mathbf{z} := \perp$                                  ▷ Reject $\mathbf{z}$ if it is unsafe.
8:     else
9:         Return $(\mathbf{z}, c)$                               ▷ Output the signature.
```

After collecting a correct signature $(\mathbf{z}, c)$, the attacker aims to inject faults (bit flips) using Rowhammer so that the signing algorithm would release such a faulty signature.

Rowhammer Target. Next, we present the target variable in Dilithium into which our attacker aims to inject bit flips. An ideal Rowhammer target should fulfill the following requirements:

- Injecting bit flips into the target should impact the computation of c such that the signature released by the signing server contains a faulty c'.
- When the target is injected with a fault, it should not impact the computation of $\mathbf{y}$, ensuring that the faulty signature released by the signing server is generated using the same $\mathbf{y}$ as the correct signature.
- The target is a static variable susceptible to tampering by the attacker using Rowhammer techniques.

After examining the Dilithium signing algorithm presented in Algorithm 7, we identify the variable ρ as our Rowhammer target. A fault injected into ρ could only contaminate c but not $\mathbf{y}$. Furthermore, ρ is not a variable computed during the Dilithium signing process; rather, it is a static public buffer persistently maintained by the signing server.

Secret Key Extraction. We now detail the secret key extraction. Recalling from Algorithm 7, Dilithium signing contains a while loop wherein a new secret nonce $\mathbf{y}$ is deterministically generated during each iteration. Suppose the correct signature $(\mathbf{z}, c)$ is generated at loop iteration t. While a fault injected into ρ does not contaminate $\mathbf{y}$, it may result in the faulty signature $(\mathbf{z}', c')$ failing to terminate at the same loop iteration t. Consequently, $(\mathbf{z}', c')$ could be generated using a different $\mathbf{y}'$, and the attacker cannot extract $\mathbf{s}_1$ as $\frac{\mathbf{z}-\mathbf{z}'}{c-c'}$ since $\mathbf{y}$ and $\mathbf{y}'$ do not cancel each other.

To address this challenge, we generate n pairs of correct and faulty signatures with n messages. The Hamming weight of $\Delta\mathbf{z} = \mathbf{z}' - \mathbf{z}$ reveals whether the correct signature $(\mathbf{z}, c)$ and the faulty signature $(\mathbf{z}', c')$ terminate at the same loop iteration. If they terminate at the same loop iteration, $\Delta\mathbf{z}$ equals to $\mathbf{s}_1 * (c' - c)$. Since $\mathbf{s}_1$, c' and c have low Hamming weight , $\Delta\mathbf{z}$ is also of low Hamming weight. On the other hand, if they terminate at different loop iterations, $\Delta\mathbf{z}$ equals to $\mathbf{s}_1 * (c' - c) + \mathbf{y}' - \mathbf{y}$. Since $\mathbf{y}'$, $\mathbf{y}$ have high Hamming weight, $\Delta\mathbf{z}$ is also of high Hamming weight. Prior work has shown that given a random pair of correct and faulty signatures with faults injected into ρ, the success rate of secret key recovery is around 14% [36].

6.3. Rowhammer

We now describe the steps taken to induce the necessary bit flips in a Dilithium signing server.

Experimental Setup. We used a setup similar to our attack on Kyber (see section 4) with a desktop running Ubuntu 22.04 and containing a single Samsung DDR4 8GiB DIMM (part number M378A1K43BB2-CRC). Using the reference implementation of Dilithium (security level 5) [37], we set up a signing server that listens to messages on localhost and signs them with its secret key. Our attack process ran with normal user privileges.

Memory Profiling. We obtain contiguous memory and find aggressor pairs using THPs as we did in the attack on BIKE-KEM (see section 5). The reference implementation packs ρ, our Rowhammer target, and the secret key into a single memory buffer where ρ is the first 32 bytes of the buffer. Our signing server maps the secret key buffer in a page aligned manner, which places ρ in the first 32 bytes of the page. In search of a bit flip with the required offset, we perform multi-bank hammering on two banks (10 aggressors each). We search for up to 20MB of THPs and exit and retry if no suitable flips are found.

Memory Massaging. We unmap the victim page (after breaking THP into normal 4KB pages) and establish a connection with the signing server. Next, the server maps the key into its address space and, since we just unmapped the vulnerable page, the secret key will be mapped to it. We then hammer the aggressors and send the messages to the server. As in the case for our attack on BIKE, no performance degradation is required for this attack.

Experimental Results. We found 396 unique bit flips before finding a target page with a suitable bit flip at index 7. We massaged the target page successfully and flipped ρ making the signing server output 10,000 faulty signatures which we used to extract the secret key.

6.4. Attack on Randomized Dilithium

A careful implementation understanding the risk of side-channel attacks may choose to use the randomized version of Dilithium to prevent the attack above. To overcome randomized signing, we now present an instruction flipping attack on randomized Dilithium. We follow the same technique presented in the attack on BIKE-KEM (see Section 5).

Instruction Flip Attack on Randomized Dilithium Listing 4 and 5 show a snippet of the Dilithium key generation function in assembly and C, respectively. Changing the last argument of the `shake256` function has the effect of making the variables `rho`, `rhoprime`, and `key` deterministic, which results in a deterministic keypair generation. To that end we target the argument of the instruction on line 5 in Listing 4 and flip it from 0x20 to 0x00 leading to a deterministic key generation.

```
1  00000000000015f0 <
       pqcrystals_dilithium5_ref_keypair>:
2     ...
3
4     1665:  e8 26 ff ff ff        call   1590 <
       randombytes>
5     166a:  b9 20 00 00 00        mov    $0x20,%ecx
6     166f:  48 89 ea              mov    %rbp,%rdx
7     1672:  48 89 ef              mov    %rbp,%rdi
8     1675:  be 80 00 00 00        mov    $0x80,%esi
9     167a:  e8 e1 77 00 00        call   8e60 <
       pqcrystals_dilithium_fips202_ref_shake256>
10
11    ...
```

Listing 4: Assembly dump snippet of Dilithium key generation

```
int crypto_sign_keypair(uint8_t *pk, uint8_t *sk)
    {
  ...
  /* Get randomness for rho, rhoprime and key */
  randombytes(seedbuf, SEEDBYTES);
  shake256(seedbuf, 2*SEEDBYTES + CRHBYTES,
    seedbuf, SEEDBYTES);
  rho = seedbuf;
  rhoprime = rho + SEEDBYTES;
  key = rhoprime + CRHBYTES;
  ...
}
```

Listing 5: Dilithium key generation Code

7. Countermeasures

In this section we discuss the possible countermeasures to our attacks and make recommendations for hardening post-quantum cryptographic implementations against Rowhammer. We split the recommendations into software implementation recommendations and system-level Rowhammer defenses.

7.1. Implementation Defenses

First, the attack on Kyber is a failure-boosting key recovery attack. To accomplish the attack, we first use Rowhammer to introduce additional noise in the key generation process which will enable higher likelihood of failing ciphertext decryption. We then create sessions with the victim, making it generate a sufficient number of failing ciphertexts, and thereby allowing us to recover the secret key. Preventing either of these two phases can deter or eliminate this attack direction, and the recommendations should hold for any failure boosting attack.

Reducing Key Exposure to Rowhammer. Our first recommendation is to limit the amount of time that key material is in a vulnerable memory location. The Kyber specification utilizes a performance improvement through translating key material to the NTT domain for multiplication. Achieving tight control on the decryption failure rate through flips in the NTT domain is untenable. Therefore, the key material was vulnerable to bit flip attacks only during random sampling, and our attack on Kyber required a significant amount of performance degradation to achieve sufficient bit flips during that window.

Reducing vulnerable time can be accomplished in a number of ways. Kyber utilized an alternative numerical representation that was resistant to bit flips. Optimized versions of these schemes may utilize vector extensions to reduce times. For implementations that keep vulnerable material in memory for longer (e.g. signature schemes), a redundant copy or hash can be created to verify that the material has not been manipulated.

Key Auditing. Our second recommendation is to audit keys to verify assumptions. In our Kyber attack, we manipulate the error matrix, which is a component of the secret key. The specification is set that the error term is composed of random values between $[-2, 2]$. Our bit flips created two components that had 256 and 64 added to their respective values. Inspecting the error term in the public key would reveal that the key had violated this assumption and should be rejected. As this component is secret, the rejection must be identified by the key owner. To publicly identify potentially manipulated keys, the public key can be used to generate a large number of ciphertexts which should have a near-zero probability of resulting in a failure. An auditing process could generate ciphertexts and reject a key which results in any number of failures. Our decryption failure rate was $\sim 1/10,000$ for this attack which is detectable with over 99.9999% accuracy when generating 100k test ciphertexts–a process which consumes approximately 10 seconds.

Memory Layout Unfriendly to Rowhammer. Next, bit-flip attacks for decryption failure attacks are reliant on sparse memory representations. The terms in the error matrix for the Kyber reference implementations were held as 16-bit integers despite representing only 5 unique values during sampling. Sampling from a centered binomial distribution to produce error terms between $[-2, 2]$ results in nearly 70% of values in the vulnerable region having 14 leading zeros. Bit packing these error values prior to NTT conversion would prevent decryption failure attacks because the error term could not exist beyond the specification. Put more simply–the majority of bits in the page were susceptible to a bit flip in a known direction to achieve the desired outcome, and any of the error terms were equally valuable, resulting in a large vulnerable memory space.

Rowhammer Code Hardening. The instruction flip attacks on BIKE and randomized Dilithium created deterministic keys by manipulating program flow. Specifically, we eliminated the entropy passed to the implementations by redirecting the output from calls to the system random number generation that initializes the pseudorandom number generator. Relying on a single call to the system RNG presents a single point of failure. Preventing this attack is largely a systems level question as cryptographic processes need to be assured of deterministic program flow. Implementations can attempt to address these challenges through validation. Victim processes shouldn't rely on single calls to system RNG. Instruction flip attacks require significant precision to prevent corrupting the process, and each call to system RNG represents potential for the attacker to fail. Additionally, implementations should check for low-entropy seeds. Implementations can check that the hamming weight of a seed passed to a pseudorandom function is above some threshold. In the case that the attacker can continue to manipulate program flow, an auditing process should check that the generated key was not generated from a low-entropy seed. These can be pre-calculated up-to some desired depth to speed the auditing process.

7.2. Rowhammer Defenses

An alternative to Rowhammer hardened software implementations is to prevent bit-flip attacks altogether. There are a number of defenses to prevent Rowhammer attacks that have been proposed in both hardware and software.

Counter-Based Rowhammer Mitigations. Counter-based defenses track the number of activations made to a DRAM row and mitigate the victims (either by refreshing or replacing) when any of it's neighboring rows reaches a certain threshold of activations [38], [39]. However, as DRAM becomes more densely packed, the threshold decreases and increases the SRAM area cost for these trackers making them prohibitively costly. Recently, researchers have proposed employing the Last-Level Cache (LLC) as an activation counter to scale for low Rowhammer thresholds and avoid prohibitive area costs [40].

Targeted Row Refresh (TRR). TRR, which is deployed in DDR4 , employs a tracking based mitigation design but cannot track more than a certain number of rows at a time due to hardware limitations. This limitation has caused it to be largely ineffective against attacks that hammer many rows at once [14].

Error Correcting Memory. Error Correcting Codes (ECC) correct bit flips when they are detected upon reading from memory. They use extra bits stored in DRAM that act as a parity checksum. Although ECC raises the barrier for executing a successful rowhammer attack, they do not guarantee protection as shown by [41].

Software Defenses. Isolating memory of different security domains into separate physical regions in memory is a proposed software based defense against rowhammer. For example, [42] isolate kernel memory to defend against userspace Rowhammer attacks against the kernel. Additionally, [43] protect Page Table Entries using a software tracking based mechanism similar to TRR. Despite claims of low performance overheads, software based mitigation has not been widely adopted.

Hardware Accelerated Cryptography. As we discussed, for our attack on Kyber we needed a sufficiently long window to perform rowhammer. We achieved such a window through performance degradation which was made possible due to the nested loops in the `polyvec_ntt` function. By flushing instructions in the deepest level of the nested loop we were able to force many instruction cache misses in our targeted hammering window.

On the other hand, if the `polyvec_ntt` function used vectorized instructions, nested loops would not be present in the implementation and the program would be less prone to performance degradation attacks.

8. Conclusion

In this paper, we have shown how three prominent post-quantum cryptography schemes are vulnerable to Rowhammer-based bit-flip attacks. We demonstrated an end-to-end key recovery using a failure-boosting decryption failure attack on Crystals-Kyber. Additionally, we demonstrated an end-to-end key recovery on deterministic Dilithium using a Rowhammer fault-injection attack, and on randomized Dilithium using an instruction flip attack. Finally, we demonstrated an end-to-end key recovery attack on BIKE by manipulating program control flow. We note that these attacks demonstrate three completely separate attack vectors using a single underlying attack primitive–namely bit-flips. In performing these attacks, we noted characteristics of the schemes that had the effect of making the attacks more difficult and gave implementation recommendations based on our findings.

Acknowledgments

Samy Amer and Daniel Genkin are supported by the Air Force Office of Scientific Research (AFOSR) under award number FA9550-24-1-0079; the Alfred P Sloan Research Fellowship; the Defense Advanced Research Projects Agency (DARPA) under contract numbers W912CG-23-C-0022,and gifts from Cisco and Qualcomm. Arkady Yerukhimovich is supported in part by NSF grants CNS-1955620 and CNS-2144798 (CAREER). Alex Nelson is supported in part by NSF Grants CNS-2213738 and IIS-2237945 (CAREER).

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

[1] P. Pessl, L. G. Bruinderink, and Y. Yarom, “To bliss-b or not to be: Attacking strongswan’s implementation of post-quantum signatures,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1843–1855.

[2] J. Bootle, C. Delaplace, T. Espitau, P.-A. Fouque, and M. Tibouchi, “Lwe without modular reduction and improved side-channel attacks against bliss,” in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2018, pp. 494–524.

[3] P. Ravi, M. P. Jhanwar, J. Howe, A. Chattopadhyay, and S. Bhasin, “Side-channel assisted existential forgery attack on dilithium-a nist pqc candidate,” *Cryptology ePrint Archive*, 2018.

[4] J. Howe, A. Khalid, M. Martinoli, F. Regazzoni, and E. Oswald, “Fault attack countermeasures for error samplers in lattice-based cryptography,” in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[5] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, “Flipping bits in memory without accessing them: An experimental study of dram disturbance errors,” in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 361–372.

[6] S. Islam, K. Mus, R. Singh, P. Schaumont, and B. Sunar, “Signature correction attack on dilithium signature scheme,” in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, 2022, pp. 647–663.

[7] M. Fahr Jr, H. Kippen, A. Kwong, T. Dang, J. Lichtinger, D. Dachman-Soled, D. Genkin, A. Nelson, R. Perlner, A. Yerukhimovich *et al.*, “When frodo flips: End-to-end key recovery on frodokem via rowhammer,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 979–993.

[8] V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, D. Stehlé, and S. Bai, “Crystals-dilithium algorithm specifications and supporting documentation (version 3.1),” *NIST submission*, 2021.

[9] NIST, “Post-quantum cryptography standardization,” Apr 2022. [Online]. Available: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization

[10] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, “CRYSTALS-Kyber (version 3.02) – submission to round 3 of the nist post-quantum project.” https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf.

[11] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, “Status report on the third round of the nist post-quantum cryptography standardization process,” 2022.

[12] N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Ghosh, S. Gueron, T. Güneysu, C. A. Melchor, R. Misoczki, E. Persichetti, J. Richter-Brockmann, N. Sendrier, J.-P. Tillich, V. Vasseur, and G. Zémor, “BIKE: Bit flipping key encapsulation,” https://bikesuite.org/files/v5.0/BIKE_Spec.2022.10.10.1.pdf.

[13] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, “CRYSTALS-Dilithium – algorithm specifications and supporting documentation (version 3.1),” https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf, [Accessed 06-06-2024].

[14] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, “TRRespass: Exploiting the Many Sides of Target Row Refresh,” in *S&P*, May 2020, best Paper Award, Pwnie Award for the Most Innovative Research, Honorable Mention in IEEE MICRO Top Picks. [Online]. Available: https://comsec.ethz.ch/wp-content/files/trrespass_sp20.pdf

[15] I. Kang, W. Wang, J. Kim, S. van Schaik, Y. Tobah, D. Genkin, A. Kwong, and Y. Yarom, “Sledgehammer: Amplifying rowhammer via bank-level parallelism.”

[16] P. Jattke, M. Wipfli, F. Solt, M. Marazzi, M. Bölcskei, and K. Razavi, “Zenhammer: Rowhammer attacks on amd zen-based platforms,” in *33rd USENIX Security Symposium (USENIX Security 2024)*, 2024.

[17] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O’Connell, W. Schoechl, and Y. Yarom, “Another flip in the wall of Rowhammer defenses,” in *IEEE SP*, 2018, pp. 245–261.

[18] D. Gruss, C. Maurice, and S. Mangard, “Rowhammer.js: A remote software-induced fault attack in JavaScript,” in *DIMVA*, 2016, pp. 300–321.

[19] M. Seaborn and T. Dullien, “Exploiting the DRAM Rowhammer bug to gain kernel privileges,” https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html, 2015.

[20] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, “Flip feng shui: Hammering a needle in the software stack,” in *USENIX Security*, 2016, pp. 1–18.

[21] M. Lipp, M. Schwarz, L. Raab, L. Lamster, M. T. Aga, C. Maurice, and D. Gruss, “Nethammer: Inducing rowhammer faults through network requests,” in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2020, pp. 710–719.

[22] K. Mus, Y. Doröz, M. C. Tol, K. Rahman, and B. Sunar, “Jolt: Recovering tls signing keys via rowhammer faults,” in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 1719–1736.

[23] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, “Rambleed: Reading bits in memory without accessing them,” in *41st IEEE Symposium on Security and Privacy (S&P)*, 2020.

[24] D. Hofheinz, K. Hövelmanns, and E. Kiltz, “A modular analysis of the Fujisaki-Okamoto transformation,” Cryptology ePrint Archive, Report 2017/604, 2017, https://eprint.iacr.org/2017/604.

[25] P. Schwabe, R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, G. Seiler, and D. Stehlé, “CRYSTALS-KYBER,” National Institute of Standards and Technology, Tech. Rep., 2020, available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[26] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé, “CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM,” Cryptology ePrint Archive, Report 2017/634, 2017, https://eprint.iacr.org/2017/634.

[27] D. Dachman-Soled, L. Ducas, H. Gong, and M. Rossi, “LWE with side information: Attacks and concrete security estimation,” in *Advances in Cryptology – CRYPTO 2020, Part II*, ser. Lecture Notes in Computer Science, D. Micciancio and T. Ristenpart, Eds., vol. 12171. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 17–21, 2020, pp. 329–358.

[28] A. C. Aldaya and B. B. Brumley, “HyperDegrade: From GHz to MHz effective CPU frequencies,” in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 2801–2818. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/aldaya

[29] S. Islam, A. Moghimi, I. Bruhns, M. Krebbel, B. Gulmezoglu, T. Eisenbarth, and B. Sunar, “SPOILER: Speculative load hazards boost rowhammer and cache attacks,” in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 621–637. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/islam

[30] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, “{DRAMA}: Exploiting {DRAM} addressing for {Cross-CPU} attacks,” in *25th USENIX security symposium (USENIX security 16)*, 2016, pp. 565–581.

[31] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, “GitHub - pq-crystals/kyber — github.com,” https://github.com/pq-crystals/kyber.

[32] Y. Yarom and K. Falkner, "{FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack," in *23rd USENIX security symposium (USENIX security 14)*, 2014, pp. 719–732.

[33] L. Ducas, N. Drucker, S. Gueron, and D. Kostic, Dusan Stehlé, "GitHub - awslabs/bike-kem — github.com," https://github.com/awslabs/bike-kem.

[34] D. Gruss, E. Kraft, T. Tiwari, M. Schwarz, A. Trachtenberg, J. Hennessey, A. Ionescu, and A. Fogh, "Page cache attacks," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 167–180.

[35] V. Lyubashevsky, "Fiat-shamir with aborts: Applications to lattice and factoring-based signatures," in *ASIACRYPT*, 2009.

[36] L. G. Bruinderink and P. Pessl, "Differential fault attacks on deterministic lattice signatures," in *CHES'18*, 2018.

[37] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, G. Seiler, P. Schwabe, and D. Stehlé, "GitHub - pq-crystals/dilithium — github.com," https://github.com/pq-crystals/dilithium, [Accessed 29-04-2024].

[38] Y. Park, W. Kwon, E. Lee, T. J. Ham, J. H. Ahn, and J. W. Lee, "Graphene: Strong yet lightweight row hammer protection," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 1–13.

[39] M. Qureshi, A. Rohan, G. Saileshwar, and P. J. Nair, "Hydra: Enabling low-overhead mitigation of row-hammer at ultra-low thresholds via hybrid tracking," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 699–710.

[40] A. Saxena and M. Qureshi, "Start: Scalable tracking for any rowhammer threshold," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 578–592.

[41] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ECC memory against Rowhammer attacks," in *IEEE SP*, 2019.

[42] F. Brasser, L. Davi, D. Gens, C. Liebchen, and A.-R. Sadeghi, "{CAn't} touch this: Software-only mitigation against rowhammer attacks targeting kernel memory," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 117–130.

[43] Z. Zhang, Y. Cheng, M. Wang, W. He, W. Wang, S. Nepal, Y. Gao, K. Li, Z. Wang, and C. Wu, "SoftTRR: Protect page tables against rowhammer attacks using software-only target row refresh," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 399–414. [Online]. Available: https://www.usenix.org/conference/atc22/presentation/zhang-zhi

Appendix A. Meta Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

A.1. Summary

This paper proposes three attacks on the implementations of recent post-quantum cryptography algorithms. The attacks are based on injecting bit flips with Rowhammer, and allow for full recovery of the secret key in practical conditions.

A.2. Scientific Contributions

- Identifies an Impactful Vulnerability
- Provides a Valuable Step Forward in an Established Field

A.3. Reasons for Acceptance

1) The PC found this work to be an effective demonstration of the many subtle techniques that must be stitched together to perform successful key recovery using Rowhammer.
2) The targeted algorithms are among the recently announced finalists and alternates of the multi-year NIST competition for post-quantum cryptography and are expected to enjoy wide uptake in the near future.

A.4. Noteworthy Concerns

1) The PC found that it is challenging to determine if the environmental conditions in the paper are realistic.
2) The PC found that some of the underlying attack techniques are well-known.