

Signature Correction Attack on Dilithium Signature Scheme

Saad Islam
Worcester Polytechnic Institute
Worcester, MA, USA
sislam@wpi.edu

Koksal Mus
Worcester Polytechnic Institute
Worcester, MA, USA
kmus@wpi.edu

Richa Singh
Worcester Polytechnic Institute
Worcester, MA, USA
rsingh7@wpi.edu

Patrick Schaumont
Worcester Polytechnic Institute
Worcester, MA, USA
pschaumont@wpi.edu

Berk Sunar
Worcester Polytechnic Institute
Worcester, MA, USA
sunar@wpi.edu

Abstract—Motivated by the rise of quantum computers, existing public-key cryptosystems are expected to be replaced by post-quantum schemes in the next decade in billions of devices. To facilitate the transition, NIST is running a standardization process which is currently in its final Round. Only three digital signature schemes are left in the competition, among which Dilithium and Falcon are the ones based on lattices. Besides security and performance, significant attention has been given to resistance against implementation attacks that target side-channel leakage or fault injection response. Classical fault attacks on signature schemes make use of pairs of faulty and correct signatures to recover the secret key which only works on deterministic schemes. To counter such attacks, Dilithium offers a randomized version which makes each signature unique, even when signing identical messages.

In this work, we introduce a novel Signature Correction Attack which not only applies to the deterministic version but also to the randomized version of Dilithium and is effective even on constant-time implementations using AVX2 instructions. The Signature Correction Attack exploits the mathematical structure of Dilithium to recover the secret key bits by using faulty signatures and the public-key. It can work for any fault mechanism which can induce single bit-flips. For demonstration, we are using Rowhammer induced faults. Thus, our attack does not require any physical access or special privileges, and hence could be also implemented on shared cloud servers. Using Rowhammer attack, we inject bit flips into the secret key s_1 of Dilithium, which results in incorrect signatures being generated by the signing algorithm. Since we can find the correct signature using our Signature Correction algorithm, we can use the difference between the correct and incorrect signatures to infer the location and value of the flipped bit without needing a correct and faulty pair. To quantify the reduction in the security level, we perform a thorough classical and quantum security analysis of Dilithium and successfully recover 1,851 bits out of 3,072 bits of secret key s_1 for security level 2. Fully recovered bits are used to reduce the dimension of the lattice whereas partially recovered coefficients are used to reduce the norm of the secret key coefficients. Further analysis for both primal and dual attacks shows that the

lattice strength against quantum attackers is reduced from 2^{128} to 2^{81} while the strength against classical attackers is reduced from 2^{141} to 2^{89} . Hence, the Signature Correction Attack may be employed to achieve a practical attack on Dilithium (security level 2) as proposed in Round 3 of the NIST post-quantum standardization process.

1. Introduction

In recent years, quantum computers have made steady progress to the point where they are considered a threat to traditional public-key cryptosystems based on the conjectured hardness of problems such as integer factorization and discrete logarithm. In a landmark result, Shor introduced an algorithm [1] that can solve the classically conjectured hard problems of factorization and discrete logarithm in polynomial time with the aid of a quantum computer. Symmetric-key systems will also be affected, albeit to a lesser extent. Using Grover's algorithm [2] one may recover symmetric keys by searching through the key-space with square-root time complexity. Hence one may overcome Grover, by equivalently doubling key lengths of symmetric schemes and output sizes of hash functions. As Key Exchange Mechanism (KEM) uses public-key schemes to exchange the symmetric keys, there is a need to develop schemes based on quantum-secure hard problems.

To aid the transition to post-quantum cryptography (PQC), the US NIST has announced a PQC standardization process in 2016 [3]. The process started with 82 submissions for public-key encryption (PKE), key encapsulation mechanisms (KEM) and digital signatures. 69 schemes were passed into Round 1, 26 were able to get into Round 2 and currently there are seven finalists and eight alternate candidates in Round 3 expected to be completed by the end of 2022. Similar schemes were merged together and some were attacked by the cryptographic community and were taken out of the competition [4], [5], [6], [7], [8], [9], [10], [11], [12]. There are five categories based on the underlying hard problems: lattice-based, code-based, hash-based, isogeny-based and multivariate schemes. These schemes offer varying key sizes under varying performance figures, but lattice-based schemes

have comparatively compact keys and exhibit better performance. Five out of seven finalists belong to the lattice category; two of the lattice schemes are digital signatures with Dilithium [13] being one of them. It belongs to the CRYSTALS family having another finalist KYBER which is a KEM. Both are based on the conjectured hard module Learning With Errors (LWE) problem.

The cryptographic community as well as companies have started integrating the finalists from the NIST competition into existing cryptographic libraries like OpenSSL. An open-source project named Open Quantum Safe (OQS) aims to support the development and prototyping of quantum-resistant cryptography [14]. PQShield [15] is providing four different products for hardware and firmware for embedded devices, SDK for mobile and server technologies and encryption solution for messaging platforms. Another company, QuSecure [16], is providing a software solution to protect the data at rest. The transition from classic to post-quantum algorithms is urgently needed to ensure forward secrecy.

According to the status report on the second Round of NIST PQC [17], evaluation is based on three criteria: 1) Security. 2) Cost and performance. 3) Algorithm and implementation characteristics. The third criterion is very important since even if a scheme is mathematically secure, it may succumb to side-channel and fault attacks targeting the implementation. Indeed, in recent years, numerous side-channel attacks e.g. [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29] and fault attacks e.g. [30], [31], [32], [33], [34], [35], [36], [37], [5], [38] have been demonstrated by the research community on PQC schemes. These include cache attacks, power and EM side-channels, EM and laser injections, clock-glitches and the Rowhammer attack. A major challenge in applying these attacks on PQC schemes, is that PQC schemes have massive key sizes (KBytes) while the attacks can reveal only a few bits per attempt. Yet, even a few revealed key bits may reduce the security strength below the level specified by the PQC standard. Another challenge for side-channel attacks is that all the finalists have constant-time AVX2 implementations for example they do not have secret dependent branches or other timing variations based upon the secret key. Also, the schemes in Round 3 have withstood more than five years of cryptanalysis by the cryptographic community and the underlying hard problems have been analyzed for decades. For the fault attacks like Differential Fault Attacks (DFA), PQC schemes already have a mitigation by randomizing the nonce values. DFA works on a principle of taking the difference of the correct and faulty pair of output and mathematically recover the secret key. After this mitigation, the same message signed or encrypted twice gives a different signature or ciphertext and the attacker is unable to collect a faulty and correct pair of the same message. The exception is the recently introduced fault attack named QuantumHammer [30] which exploits the faulty signatures to recover secret key bits. However, QuantumHammer works only on LUOV, a multivariate signature scheme eliminated in Round 3.

In this work, we are the first to demonstrate a fault attack on the randomized version of Dilithium in Round 3, which is also applicable to the deterministic version. Previous fault attacks on Dilithium [31], [32], [5] are

only applicable to the deterministic version of Dilithium in Round 1. DFA requires a pair of faulty and correct signatures which can be collected by signing the same message twice and faulting in the second iteration. To prevent this DFA, Round 2 Dilithium introduced signature randomization by using a different nonce for every signature generation. Our proposed Signature Correction attack is independent of the nonce and hence applicable to both randomized and deterministic versions of Dilithium. Bruinderink *et al.* [31] based their analysis on hypothetical faults without experimental confirmation. Ravi *et al.* [32], [5] have experimented using EM fault injection on the reference implementation for ARM-Cortex-M4. All of these attacks require physical access to induce the faults.

We propose Signature Correction attack on Dilithium and demonstrate it on the constant-time AVX2 implementation using a Rowhammer attack. Our Signature Correction attack can work with any single fault injection mechanism. We have chosen Rowhammer, because it is a software-only fault attack that can be launched remotely. Also, it has not been mitigated and can be dangerous in cloud scenarios where different users shares the same DRAM [39], [40].

1.1. Our Contribution

We introduce the Signature Correction Attack on the Dilithium signature scheme which recovers secret key bits using only the faulty signatures and the public key. The attack works by first inducing bit flips in the signing process, then collecting the faulty signatures and finally recovers the secret key bits while trying to correct the faulty signature using verification algorithm as an oracle. The faults are induced using a practical and software only Rowhammer attack to produce the faulty signatures. In summary, in this work:

- 1) We introduce the Signature Correction Attack on Dilithium signature scheme on both randomized as well as deterministic version. The Signature Correction Attack only requires faulty signatures and the public key to mathematically locate single bit faults on the secret key and to reveal the exact value of the bit-flip independent of the fault mechanism used.
- 2) We practically demonstrate the Rowhammer attack as a fault injection mechanism for Signature Correction on constant-time AVX2 implementation of Dilithium to generate the faulty signatures. Unlike physical fault mechanisms like EM, laser or clock-glitches, Rowhammer does not require any physical access which permits remote attacks on shared servers and is also applicable through JavaScript.
- 3) We recover partial secret key of 883 bits out of 3,072 bits for Dilithium security level 2 in about 2 hours of online Rowhammer attack and negligible amount of post-processing.
- 4) Careful analysis of the encoding of the secret key allows us to increase the number of recovered bits from 883 to 1,522. Additionally, analysis on the positions of the recovered bits reveal an additional 329 bits hence significantly extending

the key material. Detailed analysis is given in Section 5.

- 5) Further analysis of lattice attacks shows a much reduced security for Dilithium security level 2 below the NIST’s requirements, i.e. from 2^{128} to 2^{81} . Hence a partial key material collection and recovery with Signature Correction Attack followed by a lattice attack may indeed compromise Dilithium level 2 in practice.
- 6) Our Signature Correction Attack is applicable to all variants of Dilithium currently in Round 3 including the randomized versions recommended for side-channel and fault attacks.
- 7) We propose countermeasures to detect and prevent the Signature Correction attack by temporal and spatial redundancy techniques as well as through Rowhammer mitigations.

1.2. Outline

In Section 2, we describe a brief Background of Dilithium signature scheme and the Rowhammer attack. We explain our novel Signature Correction attack in Section 3. Section 4 includes the experimental results. Lattice attacks with complexity calculations are explained in Section 5. In Section 7, we propose countermeasures for our attack and Section 8 concludes the work.

2. Background

We first briefly explain the primitives of the Dilithium scheme. This is followed by an overview of the Rowhammer attack as we are using it as tool to demonstrate our Signature Correction attack.

2.1. CRYSTALS - Dilithium

The Cryptographic Suite for Algebraic Lattices (CRYSTALS) consists of two cryptographic schemes, Kyber [41], a KEM and Dilithium [13], a digital signature algorithm. The suite has been submitted to NIST PQC competition by the Crystals team and both the CRYSTALS are among the Round 3 finalists. These algorithms are based on hard problems over module lattices. We will only talk about Dilithium in this work. The security of Dilithium is based on two problems, namely, Learning With Errors (LWE) problem and SelfTargetMSIS problem. Dilithium is essentially based on Bai-Galbraith scheme proposed by Bai and Galbraith [42] in 2014. The design of the scheme is based on “Fiat-Shamir with Aborts” [43]. Dilithium has three security levels 2, 3 and 5 and also have AES versions instead of SHAKE for performance purposes. We shall just briefly explain the key generation, signing and verification algorithms of Dilithium scheme. We refer the reader to the original specifications for details [13].

2.1.1. Key Generation. The secret key vectors s_1 and s_2 of lengths l and k are sampled randomly from a uniform distribution. Each element of these vectors is a polynomial in the ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ and the coefficients are of size η , where $q = 2^{23} - 2^{13} + 1$ and $n = 256$.

Next, a $k \times l$ matrix \mathbf{A} is generated whose entries are also from R_q with relatively larger coefficients in range q . Then the LWE vector t is computed, part of which is kept secret as t_0 while the other part t_1 is made public. The matrix \mathbf{A} is also made public while s_1 and s_2 are kept secret. Dilithium key generation process can be seen in Algorithm 1 where it outputs pk as public key and sk as secret key. Unlike the Bai-Galbraith scheme, where the whole t was made public, Dilithium just makes t_1 public to reduce the size of the public key. The signature size however, is relatively increased by a small factor.

Algorithm 1 Dilithium Key Generation [13]

- 1: **Output:** pk - Public Key, sk - Secret Key
 - 2: $\zeta \leftarrow \{0, 1\}^{256}$
 - 3: $(\rho, \varsigma, K) \in \{0, 1\}^{256 \times 3} \leftarrow H(\zeta)$
 - 4: $(s_1, s_2) \in S_\eta^l \times S_\eta^k \leftarrow H(\varsigma)$
 - 5: $\mathbf{A} \in R_q^{k \times l} \leftarrow ExpandA(\rho)$
 - 6: $t \leftarrow \mathbf{A}s_1 + s_2$
 - 7: $(t_1, t_0) \leftarrow Power2Round_q(t, d)$
 - 8: $tr \in \{0, 1\}^{384} \leftarrow CRH(\rho || t_1)$
 - 9: **return** $(pk = (\rho, t_1), sk = (\rho, K, tr, s_1, s_2, t_0))$
-

2.1.2. Signature Generation. Dilithium signing has two modes of operation, deterministic which is the default and randomized, recommended for side-channel and fault attacks scenarios. The nonce y is generated using a seed ρ' which is either deterministic or randomized depending upon the mode of operation. The signature z is generated using the expression $z = y + c \cdot s_1$, where c is the challenge vector derived as depicted in Algorithm 2. An important part of the signing operation is the rejection sampling which checks if the signature z does not leak any secret information. The rejection sampling loop runs for approximately 4 to 7 times until a secure signature is generated. There is a rejection counter κ which is incremented in every loop to generate a different nonce y in each iteration.

2.1.3. Signature Verification. The Dilithium verification algorithm computes the challenge vector \tilde{c} and compares it to the \tilde{c} provided in the signature. Also, it checks the range of coefficients of signature z and the weight of the hint h . If all the three conditions are met, the signature is verified, otherwise rejected. The hint h is not kept secret since it is needed by the verifier to makeup for t_0 . We refer to the Dilithium specification for details [13].

2.2. Rowhammer Fault Injection Mechanism

We are using Rowhammer as a tool to inject faults. We briefly review the concept and operation of the Rowhammer attack, covering memory management, DRAM organization, address translation and applicability on cloud environments.

Every process has its own virtual address space which is divided into virtual pages, typically of size 4 KBytes. Memory Management Unit (MMU) translates the virtual addresses into physical addresses and keeps track in form of page tables. The memory controller integrated in modern processor then translates these physical addresses

Algorithm 2 Dilithium Signature Generation [13]

```
1: Input:  $sk$  - Secret Key,  $M$  - Message
2: Output:  $\sigma$  - Signature
3:  $\mathbf{A} \in R_q^{k \times l} \leftarrow \text{ExpandA}(\rho)$ 
4:  $\mu \in \{0, 1\}^{384} \leftarrow \text{CRH}(tr \parallel M)$ 
5:  $\kappa \leftarrow 0, (z, h) \leftarrow \perp$ 
6:  $\rho' \in \{0, 1\}^{384} \leftarrow \text{CRH}(K \parallel \mu)$  (or  $\rho' \leftarrow \{0, 1\}^{384}$ 
   randomized)
7: while  $(z, h) = \perp$  do
8:    $y \in S_{\gamma_1}^l \leftarrow \text{ExpandMask}(\rho', \kappa)$ 
9:    $w \leftarrow \mathbf{A}y$ 
10:   $w_1 \leftarrow \text{HighBits}_q(w, 2\gamma_2)$ 
11:   $\tilde{c} \in \{0, 1\}^{256} \leftarrow H(\mu \parallel w_1)$ 
12:   $c \in B_\tau \leftarrow \text{SampleInBall}(\tilde{c})$ 
13:   $z \leftarrow y + c \cdot s_1$ 
14:   $r_0 \leftarrow \text{LowBits}_q(w - c \cdot s_2, 2\gamma_2)$ 
15:  if  $\|z\| \geq \gamma_1 - \beta$  or  $\|r_0\|_\infty \geq \gamma_2 - \beta$  then
16:     $(z, h) \leftarrow \perp$ 
17:  else
18:     $h \leftarrow \text{MakeHint}_q(-c \cdot t_0, w - c \cdot s_2 + c \cdot t_0, 2\gamma_2)$ 
19:    if  $\|c \cdot t_0\|_\infty \geq \gamma_2$  or the # of 1's in  $h > \omega$ 
then
20:       $(z, h) \leftarrow \perp$ 
21:    end if
22:  end if
23:   $\kappa \leftarrow \kappa + l$ 
24: end while
25: return  $\sigma = (z, h, \tilde{c})$ 
```

Algorithm 3 Dilithium Signature Verification [13]

```
1: Input:  $pk$  - Public Key,  $M$  - Message,  $\sigma$  - Signature
2: Output: Verify / Reject
3:  $\mathbf{A} \in R_q^{k \times l} \leftarrow \text{ExpandA}(\rho)$ 
4:  $\mu \in \{0, 1\}^{384} \leftarrow \text{CRH}(\text{CRH}(\rho \parallel t_1 \parallel M)$ 
5:  $c \leftarrow \text{SampleInBall}(\tilde{c})$ 
6:  $w'_1 \leftarrow \text{UseHint}_q(h, \mathbf{A}z - ct_1 \cdot 2^d, 2\gamma_2)$ 
7: return  $[\|z\|_\infty < \gamma_1 - \beta]$  and  $[\tilde{c} = H(\mu \parallel w'_1)]$  and [#
   of 1's in  $h \leq \omega]$ 
```

into channels, ranks and banks inside the DRAM. This DRAM addressing varies from system to system and is not publicly disclosed for Intel CPUs, although the DRAM addressing was reverse engineered for some of the systems by Pessl *et al.* in 2016 [44]. Each bank then further consists of rows and columns sharing the same row buffer. A DRAM row consists of 64K cells and a cell is composed of a transistor and a capacitor. Data is stored in these capacitors in form of charge and interpreted as a zero or a one according to predefined threshold levels. As capacitors leak charge over time, there is a refresh mechanism to restore the charge of all the DRAM cells every 64ms.

As the DRAM manufacturers are trying to make memories more compact, these rows of cells are getting physically closer leading to disturbance errors from one DRAM row to another. If one row is accessed repeatedly, it might cause electrical interference with the neighboring row due to insufficient insulation and the cells in the neighboring row may leak faster. If the leakage is faster than the refresh frequency, the cells can not maintain their state, which may lead to bit flips. This is known as the

Rowhammer effect which was first introduced by Kim *et al.* in 2014 [45]. Using Rowhammer, an attacker with access to a row next to the victim row in DRAM is able to cause bit flips in the victim memory, even when the attacker resides in a process completely separate from the victim process. If the attacker hammers one row which causes bit flips in the neighboring row, it is called single-sided Rowhammer.

After this discovery, Seaborn *et al.* [46] introduced the double-sided Rowhammer which is far more effective than the earlier single-sided Rowhammer. In a double-sided Rowhammer, the attacker hammers two rows sandwiching the victim row, leaking the victim cells even faster. Veen *et al.* [47] in 2016 showed that it is also applicable to mobile platforms. Gruss *et al.* [48] introduced one-location hammering and achieved root access with opcode flipping in sudo binary in 2018. Gruss *et al.* [49] and Ridder *et al.* [50] have shown that Rowhammer can be applied through JavaScript remotely. Tatar *et al.* [51] and Lip *et al.* [52] have proved that it can be executed over the network. Rowhammer is also applicable in cloud environments [39], [40] and heterogeneous FPGA-CPU platforms [53]. In 2020, Kwong *et al.* [54] demonstrated that Rowhammer is not just an integrity problem but also a confidentiality problem.

There have been many efforts on Rowhammer detection [55], [56], [57], [58], [59], [60], [61], [62] and neutralization [49], [47], [63]. Gruss *et al.* [48] have shown that all of these countermeasures are ineffective. Some countermeasures require hardware modification, bootloader or BIOS update [63], [64], [65], [45], [66], [67] but they are not all implemented. Cojocar *et al.* [68] in 2019 reverse engineered the ECC memories showing that ECC countermeasure is not secure either. Another hardware countermeasure Target Row Refresh (TRR) has also been recently bypassed by Frigo *et al.* [69] using many-sided Rowhammer on DDR4 chips. The same work has been extended by Ridder *et al.* [50] to attack TRR enabled DDR4 chips from JavaScript. They claim that more than 80% of the DRAM chips in the market are still vulnerable to the Rowhammer attack.

3. Signature Correction Attack on Dilithium

To the best of our knowledge, there is no published work yet summarizing a fault attack on Dilithium which can work on randomized version of Dilithium. The randomized version randomly generates the nonce for each signing operation, which gives a different signature every time we sign the same message. Hence a standard DFA is not possible in case of randomized Dilithium as the attacker cannot recover a faulty and another correct signature for the same message for the same nonce. Our novel Signature Correction attack however is independent of the nonce, hence it is applicable to both randomized and deterministic versions of Dilithium.

The Signature Correction attack exploits the mathematical structure of Dilithium to recover the secret key bits by using just the faulty signatures and the public key. Thus the attack can be executed offline after collecting sufficiently many faulty signatures from an active fault attack. The attack is independent of the concrete fault injection technique. The only requirement is that the faults

should be single bit and induced before the signing step 13 of Algorithm 2 in secret key s_1 . First we define the attacker model and then explain the phases of our Signature Correction attack.

3.1. Attacker Model

When multiple tenants in cloud environments reside on the same server, they may share the same DRAM. The Rowhammer attack requires the attacker process and victim process to share a DRAM. The attacker process can then induce bit flips by just reading its own memory repeatedly [39], [40], [53]. Moreover, the DRAM must be vulnerable to Rowhammer attack which means that its memory cells must be susceptible to the hammering effect. Most types of DRAMs have been shown to be vulnerable in [69], [50]. We are not using HugePages for contiguous memory as most of the servers are not configured to use HugePages. We will explain how we detect contiguous memory in Section 3 as it is required for the double-sided Rowhammer to locate the neighboring rows in a DRAM bank. Also, the attacker has no knowledge of the DRAM mapping which is different for different memory controllers and DRAM configurations. The DRAM mapping maps physical addresses to actual DRAM ranks, banks, rows and columns which can be used by the attacker to co-locate with the victim in the same DRAM bank. In Section 3, we will explain how we use the row conflict side-channel for bank co-location. The attacker can induce bit flips in the secret key s_1 of Dilithium but she has no control over the position of bit flip within the s_1 . For security level 2 for example, the size of s_1 is 4 KBytes and the attacker has no knowledge of location of the bit flip within this 4 KBytes memory. Also, she has no idea of the value of the flipped bit. The attacker can just induce bit flips from her own process and is able to collect the faulty signatures from the victim. She can only use these faulty signatures along with the public parameters to recover the secret key bits.

3.2. Phases of the Signature Correction Attack

There are three phases in the Signature Correction Attack. First, we identify vulnerable memory locations called as templating. Then, we perform double-sided Rowhammer attack on the victim in the online phase and collect the faulty signatures. Finally, we post-process the faulty signatures and recover the flipped secret key bits by Signature Correction algorithm.

- 1) **Templating Phase:** In a pre-processing phase of the Rowhammer attack, the attacker will identify vulnerable memory locations. The victim needs not to be present during this phase.
- 2) **Online Phase:** In the online phase, the victim process is forced to map onto the identified vulnerable memory locations from the templating phase. Then the attacker induces bit flips inside the victim process and collects the faulty signatures generated by the victim.
- 3) **Post-processing Phase:** In this phase, the attacker uses the faulty signatures and the public

key to recover the secret key bits using the *Signature Correction algorithm*. This phase can be carried out offline and can be parallelized and run on distributed systems for performance.

We will first explain our novel Signature Correction algorithm. Next, we describe the templating and online phase of Rowhammer to practically demonstrate the fault injection.

3.3. Signature Correction Algorithm for Dilithium

Signature Correction is a way to recover the flipped secret key bits using faulty signatures. Since challenge c is public, the generated error in the signature can be used to find the position of the flipped bit in the secret key. The error in the faulty signature can be some certain multiples of c . Therefore, if we somehow correct the faulty signature, we are able to find the position of the bit-flip. The main idea of Signature Correction is to find the faulted bit in the secret key by the process of correcting the faulty signature by checking it using signature verification algorithm. The main difference between the standard DFA and Signature Correction Attack is that the attacker does not need to know the original signature. Finding the position of the flipped bit by the fault is different for every algorithm. In Algorithm 4, we explain it specifically for Dilithium.

3.3.1. How to trace back to the flipped bit using a faulty signature. s_1 is defined in S_η^l in Algorithm 1 step 4. Let $s_1 = (s_1^{(1)}, \dots, s_1^{(l)})$ in vector form where $s_1^{(i)} = \sum_{j=0}^{n-1} a_j^{(i)} x^j$ and $-\eta \leq a_j^{(i)} \leq \eta$, $1 \leq i \leq l$ and $0 \leq j \leq n-1$. In Algorithm 2 step 13, signature is generated by $z = y + c \cdot s_1$ where $c = \sum_{j=0}^{n-1} c_j x^j$ is a constant challenge vector. If one bit in s_1 is flipped before the signature generation, it faults the output signature $\bar{z} = y + c \cdot \bar{s}_1$. Then, the difference of the faulty and original signatures is $\Delta z = z + \bar{z} = c \cdot (s_1 + \bar{s}_1) = c \cdot \Delta s_1$. Since just one bit is flipped in s_1 , Δz has just one non-zero component which is $c_t \bar{a}_r^{(i)} x^{t+r}$, where $\bar{a}_r^{(i)}$ is the one bit difference, x^r is the position of the flip in s_1^i and c_t is the relevant component of the flipped bit in c . Note that, because of x^r term, c shifts to the right r times. Additionally, \bar{a}_r is a power of 2 since it is the 1-bit difference.

For instance, if the flip is in the first coefficient of s_1 , the changes in z appear at the same indices at which c is non-zero. If it is in the second coefficient of s_1 , the changes appear at the non-zero indexes of one bit shifted version of c and so on. This observation makes it possible to trace back to the faulty bit by just using the faulty signature and the public key. We can not only locate the position of the bit flip but also the value of the flipped bit because both have a unique effect on the error.

To recover the secret key bit by just using the faulty signature σ' , first we unpack the faulty signature to get the unpacked faulty signature z' and the challenge information \tilde{c} . Next we sample \tilde{c} to get the challenge vector c and copy it to a temporary variable \bar{c} as we will need to modify it. The idea is to add all n shifted versions of c in the faulty signature z' one by one and try to correct the

faulty signature. We can verify the correctness using the Dilithium verification, Algorithm 3, as an oracle. When the signature with the attempted correction verifies, we can tell that this is the index of the flipped coefficient. We can also tell the value of the flipped bit by trying both addition and subtraction of the shifted versions of c . We need to repeat this step for all of the L elements of s_1 to trace the flipped bit for any of the elements of s_1 .

This procedure works if the bit flip occurs in the LSB of the coefficients. If the flip is the second or third LSB, we need to add a *multiplier*, which is $2^{\text{bit_index}}$. This multiplier is first multiplied with the shifted version of the challenge vector \bar{c} and then added to the faulty signature z' . In the Dilithium implementation, the coefficients of s_1 are stored as `int32_t`, but the values of the coefficients range up to four bits depending upon the security level. Hence, we need to check up to three or four bits, we call this number as B . The algorithm however is capable of going further but there is no useful information on the MSB side as the remaining bits are the same as last useful LSB. So, we need to keep modifying the public challenge \bar{c} , multiply it with the *multiplier*, add it to the faulty signature z' and verify to see if the signature is corrected using the verification oracle. If the signature is correct, the algorithm returns the recovered bit of secret key s_1 as output. The algorithm needs at most $2 \times B \times L \times n$ number of verification to recover one bit of secret key. In practice however, the code breaks earlier upon finding the location. Algorithm 4 summarizes our attack.

Algorithm 4 Novel Signature Correction Algorithm for Dilithium

```

1: Input:  $\sigma'$  - Faulty Signature,  $M$  - Message,  $pk$  - Public Key
2: Output: (row, col, bit_index, value) - Recovered secret key bit
3:  $(z', h, \bar{c}) \leftarrow \text{unpack}(\sigma')$ 
4:  $c \leftarrow \text{SampleInBall}(\bar{c})$ 
5:  $\bar{c} \leftarrow c$ 
6: for  $\text{bit\_index}$  from 1 to 32 do
7:    $\text{multiplier} \leftarrow 2^{\text{bit\_index}-1}$ 
8:   for  $\text{row}$  from 1 to L do
9:     for  $\text{col}$  from 1 to N do
10:       $\bar{z}[\text{row}] \leftarrow z'[\text{row}] + \text{multiplier} \times \bar{c}$ 
11:       $\bar{\sigma} \leftarrow \text{pack}(\bar{z}, h, c)$ 
12:      if  $\text{sig\_verify}(pk, M, \bar{\sigma}) = \text{true}$  then
13:        return ( $\text{row}, \text{col}, \text{bit\_index}, 1$ )
14:      else
15:         $\bar{c} \leftarrow \text{circ\_shift\_right}(\bar{c})$ 
16:      end if
17:    end for
18:    for  $\text{col}$  from 1 to N do
19:       $\bar{z}[\text{row}] \leftarrow z'[\text{row}] - \text{multiplier} \times \bar{c}$ 
20:       $\bar{\sigma} \leftarrow \text{pack}(\bar{z}, h, c)$ 
21:      if  $\text{sig\_verify}(pk, M, \bar{\sigma}) = \text{true}$  then
22:        return ( $\text{row}, \text{col}, \text{bit\_index}, 0$ )
23:      else
24:         $\bar{c} \leftarrow \text{circ\_shift\_right}(\bar{c})$ 
25:      end if
26:    end for
27:  end for
28: end for

```

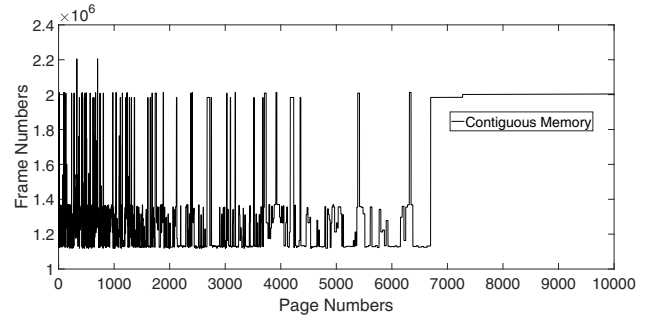


Figure 1: Contiguous memory detection. x-axis shows the page numbers of the allocated memory buffer, each page being 4 KBytes. On y-axis are the frame numbers of these pages in integer form. The straight line shows a linear increase in frame numbers; it is not a horizontal line.

3.4. Templating Phase

Signature Correction attack needs a fault mechanism which can provide faulty signatures. We are practically inducing faults using Rowhammer, a software-only technique which does not require any physical access to the target machine. Recent research has shown that it can be applied over the network [51], [52] and even remotely through JavaScript [49], [50]. There is no effective countermeasure to prevent Rowhammer completely in DRAM chips so far. Recent research has demonstrated that it is possible to apply Rowhammer even on DDR4 memories with TRR [69] mitigation as well as on ECC memories [68]. Templating phase involves three steps: contiguous memory detection, bank co-location and double-sided hammering.

3.4.1. Contiguous Memory Detection. For a double-sided Rowhammer, the attacker needs to allocate the rows exactly one above and one below around the victim in the actual DRAM. For this purpose, contiguous memory is a requirement for double-sided Rowhammer. It can be achieved using Huge-pages but that requires special configuration and privileges. We achieve contiguous memory detection using SPOILER [70] from normal user space without any special privileges. When the spoiler peaks become equally distant apart, the physical addresses become contiguous. Figure 1 shows the frame numbers of memory pages inside a buffer. We can see the contiguous memory where the frame numbers are linearly increasing. A detailed description of this approach can be found in [70].

3.4.2. Bank Co-location. A DRAM is organized in banks and every bank has a row buffer. Rowhammer attack works when both the attacker and the victim are sharing the same bank. To find the virtual addresses mapping to the same bank, we use a side-channel which is based on the row conflict. When two addresses from the same bank are accessed, it takes longer as compared to the accesses from different banks. This is because one row loaded inside the row buffer needs to be written back to its original position before loading another row. The CPU cycles taken for accessing one address and the remaining are shown in Figure 2. Depending on the maximum values of

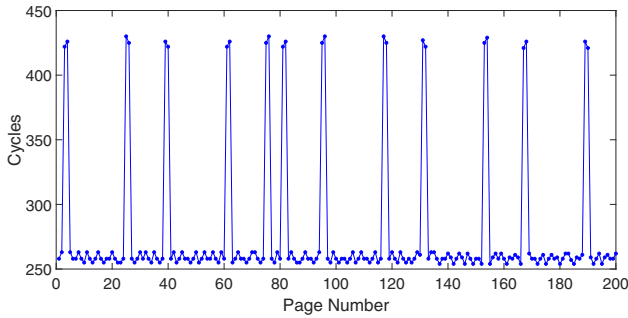


Figure 2: When two DRAM rows are accessed which reside in the same bank, we get a peak due to the row conflict. A threshold can be set to separate these rows using this side-channel information. In our experiments we have set the `THRESHOLD_ROW_CONFLICT` value as 380 cycles.

Listing 1: Typical Rowhammer instruction sequence [40]

```

loop :
  movzx rax, BYTE PTR [rcx]
  movzx rax, BYTE PTR [rdx]
  cflush BYTE PTR [rcx]
  cflush BYTE PTR [rdx]
  mfence
  jmp loop

```

the peaks, we can set a threshold to extract the addresses mapped to the same bank.

3.4.3. Double-sided Hammering. Once we identify the contiguous memory within a bank, we can start taking three rows at a time from this memory and apply double-sided Rowhammer on them. We hammer the top and bottom row and expect the bit flips in the middle row. In our experiments, we have kept the number of hammers equal to 10^6 . While keeping the record of the vulnerable rows, we keep moving onto the next three rows until for our identified contiguous memory. We have used the typical Rowhammer instruction sequences without the `mfence` as shown in Listing 1. Without the `mfence`, the number of bit flips are more as compared to with `mfence`. This is because the DRAM accesses become faster which results in quicker leakage of the charge stored in the memory cells. The number of flips with and without `mfence` are compared in Figure 3. The number of CPU cycles and the time taken by one Rowhammer instruction sequence is given in Table 1.

3.5. Online Phase

As the Rowhammer attack is highly reproducible, we first place the victim into our target vulnerable location inside the memory and repeat the double-sided Rowhammer attack by hammering the neighboring addresses. This induces the bit flips in the actual victim, and faulty signatures are produced by the victim in response. The online phase consists of two steps, first is the victim placement and the second is the double-sided Rowhammer.

3.5.1. Victim Placement. Once the attacker finds vulnerable DRAM rows, it frees the row using `munmap`. Now

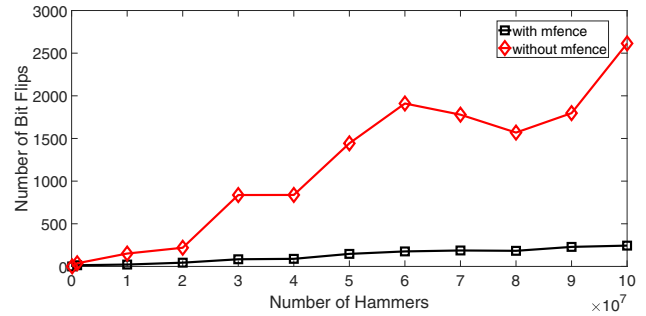


Figure 3: The number of bit flips in 1 MByte of memory in a DRAM bank out of the 8 MBytes contiguous chunk spread across 8 banks as a function of the number of hammers. The number of bit flips increases with the number of hammers and without `mfence` sequence gives much more bit flips. Approximately 0.03% of the DRAM cells are found to be vulnerable to Rowhammer attack on the DRAM model we profiled.

TABLE 1: CPU cycles and time taken by a typical Rowhammer instruction sequence on our platform.

Instruction Sequence (mV)	CPU Cycles	Time (μs)
With <code>mfence</code>	635	0.18
Without <code>mfence</code>	480	0.14

it can either wait for the victim page to take that space in the memory or use standard techniques like spraying [49], [46], [39], grooming [47] or memory waylaying [48], [54], [71] to force the victim to come at the target address. We achieve this by repeatedly mapping the secret key s_1 of the victim until it lands to the target page as shown in Figure 4. The physical addresses are checked using the `pagemap` file.

3.5.2. Double-sided hammering. When the victim is mapped to the attacker’s desired vulnerable memory location, the attacker can now apply the double-sided Rowhammer again. While the victim is signing the messages, the attacker now hammers the same rows which she found in the offline phase but this time it flips the bits in the victim process. This is because of the fact that Rowhammer effect is highly reproducible which means if you have found the vulnerable cells once, their values can be flipped again later. Finally, the victim starts producing the faulty signatures due to the bit flips in the secret key which are collected by the attacker.

When a bit is flipped on the MSB side of s_1 , it is likely that the rejection sampling condition in step 15 of Algorithm 2 repeatedly becomes true or takes too many iterations to output a faulty signature. This can create a *denial of service* scenario and can cause the victim to stuck in a loop and never output a signature unless the victim is moved to another memory location in the DRAM, making our attack harder. To counter this situation, we have set a limit on κ in Algorithm 2 to prevent the victim from going into an infinite loop. However, if there is a side-channel attack running in parallel is collecting side information, this scenario can be useful as the nonce y is changing in each iteration.

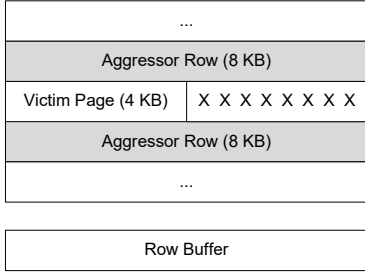


Figure 4: Victim placement and double-sided Rowhammer. To flip the bits from $1 \rightarrow 0$ inside the victim page, the attacker rows are needed to be filled with all zeros and for $0 \rightarrow 1$ flips, the attacker rows must be filled with all ones. Empirically, cells which flip both ways are very rare. Hence, a $0 \rightarrow 1$ flip may not happen in a $1 \rightarrow 0$ bad cell and vice versa.

4. Experimental Results

In this Section, first we mention our Rowhammer experimental setup and then mention the results of our Signature Correction attack experiments¹.

4.1. Experimental Setup

All the Rowhammer experiments are performed on a Haswell system with Intel(R) Core(TM) i7-4770 CPU @ 3.4GHz with 2 GBytes Samsung DDR3 part number M378B5773DH0-CH9. We have used Haswell because the AVX2 support start from Haswell and it also supports DDR3 memories. Our underlying operating system is Ubuntu 16.04 LTS.

We have performed all the post-processing on a Skylake system with Intel(R) Core(TM) i5-6440HQ CPU @ 2.60GHz having 8 GBytes DDR4 memory running Ubuntu 16.04 LTS using only a single core. The post-processing performance can be improved using multicore, GPUs or distributed computing.

4.2. Key recovery with Signature Correction Attack

We have successfully applied Rowhammer on s_1 of size 1024×32 bits for the AVX2 implementation of the Dilithium security level 2. After collecting 6,853 single-bit faulty signatures in 2.19 hours of online Rowhammer attack, we have recovered 3,735 unique bits of secret key s_1 using our Signature Correction algorithm as shown in Figure 5. Note that, the faults we can inject are far from uniform. In fact, there are locations that are unflippable. The spatial bias is highly dependent on the technology of the DRAM. In our target DRAM (M378B5773DH0-CH9), we observed heavy spatial correlations (dark vertical stripes in Figure 5). Also rejection sampling prevents faulty signatures with flips at higher locations to be released. Hence, even if we force s_1 to relocate in memory as explained in Section 3.5.1, this does not allow recovery of all s_1 bits. We start recovering the same key bits and

1. The source code for Signature Correction Attack is made available at <http://github.com/VernamLab/SignatureCorrection>.

TABLE 2: Post computation times for Signature Correction attack on a single CPU. These offline computations can be performed on a distributed system or GPUs for performance improvement.

AVX2 Implementations	Average CPU Cycles (1 Verification)	Time (Sec) (1 Signature Correction)
dilithium2	36595	0.094
dilithium3	70397	0.267
dilithium5	67719	0.263
dilithium2-AES	28901	0.071
dilithium3-AES	47614	0.177
dilithium5-AES	49479	0.200

while others that wander through unflippable locations are never recovered. Therefore, we stop the online phase and do post-processing after all flippy locations are recovered. Among the 3,735 recovered bits, 2,454 are the 0's (green pixels) and 1,281 are the 1's (red pixels). Each sub-figure represents an element (polynomial) of s_1 up to $l = 4$ for Dilithium security level 2. Each polynomial has 256 coefficients on y-axis and 32 bits per coefficient on the x-axis. Every faulty signature gives one bit of secret key. The difference of 3,118 bits is because of the repetition of the faults at the same memory location as the attacker has no control over the locations within the s_1 . 883 out of these 3,735 bits reside in the first three LSBs which should contain the actual key information. The rest of the bits from bit 4 to bit 32 are redundant, same as bit 3.

However, as the remaining bits from bit 4 to bit 32 are all same as bit 3 for each coefficient, if any of the bits are recovered from this region, we can consider it a bit recovery for LSB 3. This increases our useful bit recovery number significantly from 883 to 1,522 bits. Finally, we can say that by analyzing the positions of recovered bits in the coefficient, we can increase the number of recovered bits from 1,522 to 1,851, see Section 5.2 and Section 5.3 for details. As a summary, we have successfully recovered 1,851 bits out of the total 3,072 bits of s_1 , 3-bits each of 1024 coefficients. The results and distribution of recovered bits up to the secret key coefficients is provided in Table 8.

Table 2 shows the offline computation time needed to trace one bit of secret key for all the variants of Dilithium. These timings are for the worst case scenario of $2 \times B \times L \times n$ verification as explained in Section 3. The search is however stopped earlier once a bit is located. We have computed the post-computation times for all variants but demonstrated the Rowhammer attack on only Dilithium security level 2. However our Signature Correction attack is applicable to all variants, modes and security levels of Dilithium Round 3, where modes are randomized and deterministic, variants are SHAKE and AES and the security levels 2, 3 and 5.

5. Estimating the Diminished Security Level of Dilithium

5.1. Lattice Security with Reduced Dimension

The Signature Correction Attack can be used iteratively to recover the secret key-bits. There are however two caveats in applying Signature Correction in practice:

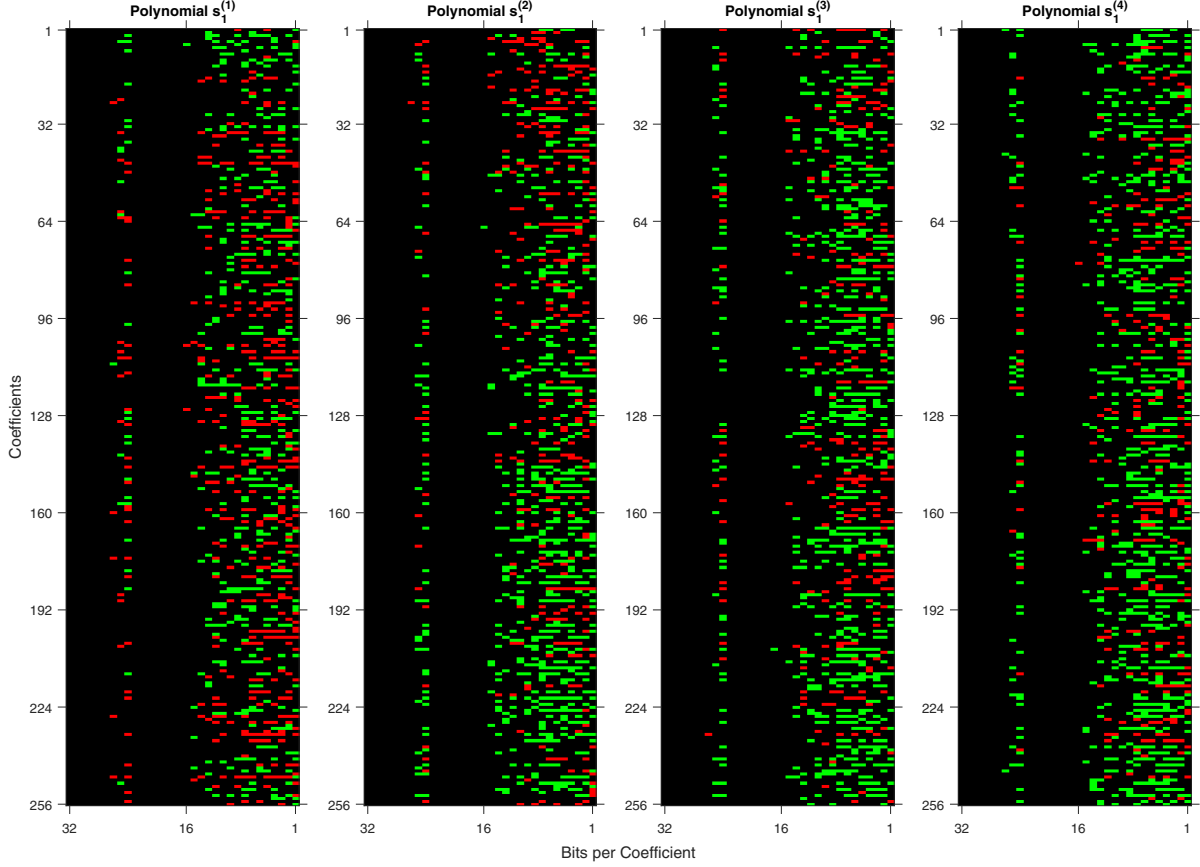


Figure 5: Recovered bits of secret key s_1 for Dilithium (security level 2). 3,735 in total with 2,454 0's (green pixels) and 1,281 1's (red pixels).

- Each Signature Correction recovers only one secret key bit. For full-key recovery we need at least 1024×3 unique faulty signatures which is rather time-consuming.
- As described below in practice we inject faults using Rowhammer, which prevents precise targeting of bits. Thus, we need many more Signature Correction iterations (and time consuming page re-allocations) in practice.

To overcome both problems, we instead opt to recover only a fraction of the key-bits to diminish the security level of Dilithium to a point where the remaining key bits can be recovered using lattice attacks.

Here we estimate the new security level of Dilithium by exploiting the recovered bits by Signature Correction attack. Briefly, Dilithium is based on the hardness of the MLWE and MSIS problems under the Strong Unforgeability under Chosen Message Attack (SUF-CMA) model. We follow the cost estimation approach of [72], [13], i.e., the MLWE problem is analyzed as an LWE problem and the security level is estimated using standard lattice hardness estimation techniques. Specifically, we base our estimate on the so-called primal and dual attacks [73], [74] and use BKZ for lattice reduction. Cost estimation of the attacks are given in [72], [13]. Note that these estimates ignore SVP oracle calls. Instead, core-SVP hardness which is the

cost of one call to an SVP oracle in dimension b is taken into account. For Quantum attacks, the Sieve algorithm is used to estimate the core hardness of underlying open problem. For the quantum sieve algorithm the heuristic complexity is $\sqrt{3/2}^{b+O(b)} \approx 2^{0.292b}$ [75], [76]. Grover's quantum search algorithm reduces the complexity down to $2^{0.265b}$ [77], [78]. Cost of solving SVP in classical attack bound is $2^{0.2075b} \approx 2^{39}$ for the best-known algorithm [73].

The Signature Correction attack allows us to recover certain number of bits of the private key s_1 . By analyzing the distribution of the recovered bits, we can recover additional bits. We follow the general methodology from [72], [79] to analyze the reduced security of Dilithium with side information recovered using the Signature Correction attack as described in Section 3. The reduced security levels can be determined by following the analysis in [13], [72], [79]. The analysis converts the equation system into an LWE instance of dimensions $256 \cdot l$ and $256 \cdot k$ by taking the coefficients of polynomial elements in the MLWE problem as the vectors of coefficients in LWE problem. Hence, the problem is reduced to finding the coefficient vectors

$$s_1, s_2 \in (\mathbb{Z}^{256 \cdot l} \times \mathbb{Z}^{256 \cdot k})$$

from \bar{A} and coefficient vector of t . Here $\bar{A} \in \mathbb{Z}_q^{256 \cdot k \times 256 \cdot l}$ is obtained by replacing all entries $a_{ij} \in R_q$ of A by

the rotation of the coefficient vectors of a_{ij} . One can show that the private key coefficients recovered using the Signature Correction Attack can be used to reduce the dimension n of the lattice formed by the equation system

$$As_1 + s_2 = t \quad (1)$$

by inserting the recovered coefficients of the secret key into its polynomial form. Moreover, inserting the recovered bits which are not used to find the coefficients can also reduce the norm of the coefficient vector. The security estimates for the scheme reduced for a given number of coefficients recovered using Signature Correction is given in Table 9. From the table, we deduce that recovering 8 coefficients of secret key reduces the attack complexity to around half of the overall complexity on average. In other words, recovering approximately 320 coefficients by Signature Correction attack is enough to reduce the attack complexity to a practical level, i.e. 80 bits. Note that, we take the norm of secret key coefficients as $\zeta = \sqrt{10}$. Estimated time to recover is given in Section 4.2.²

To recover 320 coefficients, we need 960 bits in the same coefficients. Therefore, we cannot conclude that any 960-bit recovery is enough to break the scheme since we do not have any control on the location of the recovered bits.

5.2. Exploiting the Redundant Encoding to Recover More Coefficients

In this section, we focus on how we can use the recovered bits in the most effective way to diminish the security level of Dilithium. For this purpose, we divide the coefficients of the secret key polynomial into 3 groups:

- **Group 1** has the fully recovered coefficients, i.e, 3 out of 3 bits are known in the coefficients. Number of recovered coefficients in Group 1 can directly be used to **reduce the dimension** n of the LWE system (Section 5.2).
- **Group 2** consists the coefficients in which 1 or 2 out of 3 bits are known in each coefficient. The recovered bits in this category fall short in reducing the LWE dimension further, yet they can still be used to **reduce the norm** of the secret key coefficients, i.e, unique SVP solution in LWE system (Section 5.3).
- **Group 3** is the collection of coefficients with no recovered bits. hence yielding no information about the coefficients.

When we estimate the security level, our calculations are based on the number of recovered coefficients and the norm of the remaining unknown coefficients. Secret key is defined as an l dimensional vector of n^{th} degree polynomials with coefficients in the range $[-\eta, \eta]$. In our experiments, we consider Dilithium security level 2 in which parameters are set to $\eta = 2$, $l = 4$ and $n = 256$ [13], i.e., Each coefficient of the secret key is in $\{-2, -1, 0, 1, 2\}$ but is encoded in the reference implementation as 32-bit words $\{1 \dots 1110, 1 \dots 1111, 0 \dots 0000, 0 \dots 0001, 0 \dots 0010\}$,

2. The script "scripts/PQsecurity.py" which estimates the cost of primal and dual attacks can be found at [72].

TABLE 3: Recovering an additional bit by using recovered 2-bit info by Rowhammer. Shaded rows has the additional bit recovery, i.e., full coefficient is recovered by 2-bit info.

Known bits of xyz	Possible coefficients	# of possible coefficients
00z	00z	2
01z	010	1
10z	N/A	0
11z	11z	2
0y0	0y0	2
0y1	001	1
1y0	110	1
1y1	111	1
x00	000	1
x01	001	1
x10	x10	2
x11	111	1

TABLE 4: Number of additional full coefficient recoveries by 2-bit info. Highlighted bit shows the additional bit recovery.

xyz	Rec Coeffs	$s_1^{(1)}$	$s_1^{(2)}$	$s_1^{(3)}$	$s_1^{(4)}$	Total
01z	010	10	11	6	5	32
0y1	001	12	5	6	8	31
1y0	110	7	7	6	9	29
1y1	111	10	9	5	11	35
x00	000	1	0	0	0	1
x01	001	0	0	0	0	0
x11	111	0	0	0	1	1
Total # of Rec Coeffs		40	32	23	34	129

respectively. Therefore we have a highly redundant representation, where the per coefficient entropy of secret key encoding is only 2.25 bits (in 32 logical bits). The recovered bits are distributed over the last three bits of 1024 coefficients given in Figure 5. Additionally, distribution of number of recovered bits up to the coefficients is given in Table 5.

Even though the recovered 1,522 bits are expected to give us information for about 507 coefficients, just 99 coefficients fully recovered, since only 297 out of 1,522 bits are concentrated in 99 coefficients. The remaining 1,423 bits are distributed over the remaining 857 different coefficients. On the other hand, 2-bit recovered in any coefficient yields either 0 or 1 bits of information on a coefficient as summarized in Table 3. Here coefficients are represented by xyz where z denotes the least significant bit (LSB) of the coefficient, and x represents the most significant bit (MSB) if we represent the coefficients by the last three bits. All higher order bits will be identical to x , i.e. the sign bit of the coefficient. In certain cases, with

TABLE 5: Distribution of 1,522 bits recovered by Signature Correction Algorithm to the # secret key in polynomial coefficients. Total of 99 coefficients are recovered with another 857 coefficients yielding only partial information.

	$s_1^{(1)}$	$s_1^{(2)}$	$s_1^{(3)}$	$s_1^{(4)}$	#bits	#coefs
No recovery	19	14	18	17	0	68
1 bit rec	122	126	131	110	489	489
2 bits rec	95	89	86	98	736	368
Full rec	20	27	21	31	297	99
Total	372	385	366	399	1522	1024

TABLE 6: Recovering an additional bit by using 1-bit recovered by Rowhammer. Shaded rows yield an extra bit.

Known bit of xyz	Possible coefficients	# of possible coefficients
1yz	11z	2
0yz	00z or 010	3
x1z	11z or 010	3
x0z	00z	2
xy1	001 or 111	2
xy0	x10 or 000	3

TABLE 7: Number of additional bit recovery by 1-bit info. Highlighted bit shows the recovered bit.

xyz	Rec Coeffs	$s_1^{(1)}$	$s_1^{(2)}$	$s_1^{(3)}$	$s_1^{(4)}$	Total
1yz	11z	51	46	56	37	190
x0z	00z	2	1	2	0	5
xy1	xx1	2	1	1	1	5
Total # of Rec bits		55	48	59	38	200

a 2-bit information of a coefficient we can recover the full 3-bit coefficient as shown in Table 3. For instance, if we recovered the first two bits as in the case of 01z, then due to the encoding the only possible value z can take is 0. We can fully recover a coefficient from 2-bits of information in 7 out of 12 cases as shown in the shaded rows in Table 3. With this approach, we managed to recover an additional 129 coefficients of the secret key as summarized in Table 4. The total number of recovered coefficients is increased significantly, i.e. from 99 to 228. You can find the number of recovered coefficients in Table 8.

5.3. Reducing the Norm of the Coefficients

In cases where we recover 1-bit out of a coefficient the information is not sufficient to recover the entire coefficient. However, we can still gain information useful in reducing the attack complexity. Specifically we can reduce the norm of the target vector by removing known bits from it. This reduces the complexity of the lattice search problem.

Further in certain cases the 1-bit knowledge may facilitate recovery of an additional bit of the coefficient. In Table 6, these special cases are shown in shaded rows. Analyzing the experimentally recovered bits gives us 200 of these special cases, i.e., two bits of 200 coefficients are recovered by 1-bit information. In Table 7, the number of coefficients in which extra bit recovery is possible is shown. By analyzing the recovered bits, we recovered 1 bit of 289 coefficients and 2 bits of 439 coefficients. There are 68 coefficients that we have no extra information about. Number of recovered bits and coefficients by Rowhammer and extra bit recovery method is given in Table 8. When we insert these recovered bits into the Lattice formulation the norm of secret key coefficients in the reduced system is decreased to

$$\zeta = \frac{68}{796} \times 3 + \frac{289}{796} \times 2 + \frac{439}{796} \times 1 = 1.53392.$$

By analyzing the encoding (Section 4.2), we increased the number of recovered bits from 883 to 1,522. This was achieved by taking recovered bits from 4 to 32 as the sign bit, i.e. x . Then we further increased from 1,522 to

1,851 by considering the positions in the recovered bits in the last 3 bits of the coefficient. In total, the number of fully recovered coefficients are increased from 99 to 228, and the number of coefficients with 2 bits known are increased from 368 to 439. By analyzing the encoding, we partially or fully recovered 956 coefficients of 1024 secret key coefficients, in total. The breakdown is given in Table 8.

The diminished security level of Dilithium with the fully recovered coefficients (reduced dimension \bar{n}) and reduced average norm ζ is listed in Table 9. Note that with the fully recovered coefficients the reduced security level is 124-bits for classical and 112-bits for quantum attackers. By also exploiting the encoding to increase the fully recovered coefficients from 99 to 228 and partially recovered coefficients to reduce the norm from $\zeta = \sqrt{10}$ to $\zeta = 1.53392$, we managed to significantly degrade the security level: **89-bits (classical) and 81-bits (quantum)**.

6. Discussion

6.1. Is the weakness inherent to Dilithium?

In our attack we exploited the linear structure of Step 13 in the Dilithium Signing Algorithm:

$$z \leftarrow y + c \cdot s_1.$$

To this end, we compute and check possible fault patterns as they would appear as additive terms in the faulty signature \bar{z} . This approach is enabled by the *linearly additive* secret mask y and the publicly known challenge vector c . Clearly, the presented signature correction algorithm is specific to Dilithium. However, we have also tried to produce a similar technique in the GemSS [80] and Rainbow [81] schemes which gave insufficient partial information. While the approach is generic, the particulars of the signing algorithm may still make it hard to trace the fault to the output without causing the search space to grow exponentially, thus preventing efficient signature correction. While the presented attack utilizes faulty signatures to recover secret key bits we have also exploited the highly redundant encoding of the coefficients to gain significant advantage in reducing the security level of Dilithium. This weakness is not rooted in the algorithm itself, but rather due to the choice of representation used in the implementation.

6.2. Further Reducing the Attack Complexity

Dachman-Soled et al. [6] introduced a framework for cryptanalysis of lattice based schemes when side-information in the form of “hints” about the secret and/or error is available. The framework allows the primal lattice reduction attack and allows progressive integration of hints before running a lattice reduction step. What we refer to as “recovered coefficient” and “partially recovered coefficient” correspond to “Modular hints” and “Approximate hints”, respectively. Along with the framework the authors introduced techniques for progressively sparsifying the lattice, projecting onto and intersecting with hyperplanes, and/or altering the distribution of the secret vector. One may apply these more advanced techniques to gain advantage and further degrade the security level.

TABLE 8: Recovered Information by Signature Correction up to the number of coefficients. Highlighted rows show the number of coefficients with additional bit recovery.

	$s_1^{(1)}$	$s_1^{(2)}$	$s_1^{(3)}$	$s_1^{(4)}$	#coefs
Group 3: Coefficients with no bit recovery. 68 coefficients in total.					
No recovery	19	14	18	17	68
Group 2: Coefficients with 1 bit recovery. 289 bits in 289 coefficients in total.					
1 bit rec by 1 bit	67	78	72	72	289
Group 2: Coefficients with 2 bit recovery. 878 bits in 439 coefficients in total.					
2 bits rec by 1 bit	55	48	59	38	200
2 bits rec by 2 bits	55	57	63	64	239
Group 1: Full coefficient recovery. 684 bits in 228 coefficients in total.					
Full Coef rec by 2 bits	40	32	23	34	129
Full Coefs rec by 3 bits	20	27	21	31	99
Total#recbits(1851)	467	465	448	471	1024

TABLE 9: The reduced security level of Dilithium using the Signature Correction Attack. The value \bar{n} denotes the reduced lattice dimension, b the block dimension of BKZ, and m the number of samples. Cost is given in log base 2 and is the smallest cost for all possible choices of m and b . Shaded rows show improvements: 124-bits (classical) and 112-bits (quantum) with plain Signature Correction , 89-bits (classical) and 81-bits (quantum) by also exploiting the encoding in addition to Signature Correction .

Dilithium Security Level II (128 bit) parameters: $q = 2^{23} - 2^{13} + 1, n = 1024$											
#Rec coeffs	\bar{n}	ζ	Primal Attack				Dual Attack				
			m	b	Classical	Quantum	m	b	Classical	Quantum	
0	1024	$\zeta = \sqrt{10}$	1090	485	141	128	1089	484	141	128	
0	1024	$\zeta = 1.53392$	1001	429	125	113	1027	428	125	113	
Reduced Complexities with # Recovered coefficients and Reduced Norm											
1	1023	$\zeta = \sqrt{10}$	1129	484	141	128	1132	483	141	128	
2	1022	$\zeta = \sqrt{10}$	1075	484	141	128	1074	483	141	128	
4	1020	$\zeta = \sqrt{10}$	1062	483	141	128	1062	482	140	127	
8	1016	$\zeta = \sqrt{10}$	1089	480	140	127	1090	479	140	127	
64	960	$\zeta = \sqrt{10}$	1025	446	130	118	1037	445	130	118	
99	925	$\zeta = \sqrt{10}$	981	425	124	112	997	424	124	112	
128	896	$\zeta = \sqrt{10}$	933	408	119	108	947	407	119	107	
192	832	$\zeta = \sqrt{10}$	919	369	107	97	885	369	107	97	
228	796	$\zeta = \sqrt{10}$	863	348	101	92	843	348	101	92	
288	736	$\zeta = \sqrt{10}$	799	313	91	83	788	313	91	83	
320	704	$\zeta = \sqrt{10}$	744	295	86	78	810	294	86	78	
352	672	$\zeta = \sqrt{10}$	745	276	80	73	742	276	80	73	
99	925	$\zeta = 1.53392$	902	375	109	99	957	374	109	99	
228	796	$\zeta = 1.53392$	782	306	89	81	773	306	89	81	

7. Countermeasures

Every novel attack sheds light onto how to strengthen a cryptographic scheme, and in this perspective, a discussion on countermeasures is very important. We can find considerable work on countermeasures against fault attacks on PQC schemes [31], [32], [36], [35]. In particular, Bindel *et al.* [37] have written an exhaustive literature review on countermeasures for fault attacks on lattice-based signature schemes.

For our Signature Correction attack, there are two ways to detect and prevent the fault attack. First, we can prevent or detect the fault injection mechanism, which means that we would prevent or detect Rowhammer faults. Second, we can prevent or detect the exploitation of an injected fault. This requires an algorithmic countermeasure, such as preventing faulty signatures from being returned by the signer. Algorithmic countermeasures are required because our attack is independent of the fault mechanism

used. In the following, we discuss the Rowhammer countermeasures followed by algorithmic countermeasures. Then, we provide a literature review of countermeasures against implementation attacks on lattice-based signature and encryption schemes in Table 10. In this table, we have shown countermeasures which work against timing, cache and fault attacks with a green tick mark and which don't work with a red cross mark. We show that post-quantum schemes are broadly vulnerable to three kinds of fault attacks, DFA, Instruction Skip and single-bit trace analysis. The table describes how countermeasures help against these known attacks, which include an attack on an older round-2 PQ scheme [30] as well as our proposed Signature Correction .

7.1. Rowhammer Countermeasures

We discuss two approaches to counter Rowhammer attack. One technique detects the Rowhammer attack

TABLE 10: An Overview of Countermeasures against Implementation Attacks on Lattice-Based Post-Quantum Cryptography; ✓ Countermeasure works, ✗ Countermeasure doesn't work

Countermeasures	Implementation Attacks					
	Timing [82], [83], [84]	Cache [18], [19], [82], [85]	Fault			
			DFA [86], [31]	Instruction Skip [5], [31], [32], [35], [36], [87], [88], [89]	QuantumHammer [30] (LUOV Round2)	Signature Correction Attack (this work)
Constant run-time & data-oblivious accesses [18]	✓	✓	✗	✗	✗	✗
Key-independent control flow & memory accesses [82]	✗	✓	✗	✗	✗	✗
Nonce Randomization [31], [32]	✗	✗	✓	✓	✗	✗
Temporal Redundancy [86], [32], [90]	✗	✗	✓	✓	✗	✗
Spatial Redundancy [86], [90]	✗	✗	✓	✗	✓	✓
Verify-after-sign [86], [32]	✗	✗	✓	✓	✓	✓
HPC [91]	✗	✗	✗	✗	✓	✓
DRAM Refresh Rate [92]	✗	✗	✗	✗	✓	✓

through hardware monitors, while the second technique prevents Rowhammer from happening in the first place.

Hardware Performance Counters (HPC). HPCs are special purpose registers which store the hardware events inside the CPU like cache hits and cache missed. As the Rowhammer bypasses the cache and directly hits the DRAM, there will be a significant increase in the number of cache misses which can be used to detect the Rowhammer attack. These HPCs, when paired with machine learning techniques, can detect Rowhammer attack with high accuracy. Gulmezoglu *et al.* [91] have shown an accuracy of 100% using the performance counter *LLC_Miss*.

Increasing DRAM Refresh Rate. DDR3 and DDR4 specifications require that each DRAM row must be refreshed after at least 64ms to retain its values [93]. However, as we have seen this refresh rate is not sufficient in Rowhammer scenarios where hammering the neighboring rows cause the cells to leak faster than normal and are unable to retain their charge. So, an immediate mitigation can be to decrease the refresh interval to 32ms or 16ms. Many systems allow this configuration from the BIOS for better memory stability. However, there are two downsides for this approach. The first one is that the power consumption will increase and the second one is the reduction of data transfer rate. This is because while the cells are refreshed, the data can not be read or written. Also, the Rowhammer can not be fully mitigated by this countermeasure. At most, one can significantly reduce the chances of getting bit flips. Mutlu *et al.* [92] have shown that to fully mitigate Rowhammer using the refresh rate, one needs to set the refresh rate as 8.2 ms which is 7.8 times lower than 64 ms. This will cause significant burden on power consumption and quality of service which researchers are already trying to improve [94], [95].

7.2. Algorithmic Countermeasures

Here, we discuss algorithmic countermeasures for PQC signature schemes, specifically Dilithium against

general fault attacks as well as our Signature Correction attack. These countermeasures include adding randomization, temporal and spatial redundancy techniques and verify-after-sign approach.

Randomized Signing. Due to the fault attacks based on determinism like [31], [32], Dilithium added this mitigation in Round 2 for DFAs as listed in step 6 of Algorithm 2. Here, the nonce y is generated randomly instead of generating using the message M , recommended for side-channel and fault attacks [13]. The idea is that if the attacker can not collect the faulty and correct signature pair for the same message, the DFA will not work. However, this mitigation will not work for our Signature Correction attack as it is independent of the nonce y . Whatever the value of the y is, we get the same error in the faulty signature depending upon the position of the fault in secret key s_1 .

Temporal Redundancy. Temporal Redundancy requires re-execution of same task after a certain amount of time and comparing their results. If they don't match then the output of the algorithm is disabled in order to prevent the attacker from accessing any information from the faulty signature. It makes harder for the attacker to inject the same fault in redundant computations. However, as the Rowhammer attack faults the memory and can induce permanent faults, if multiple signatures are generated using the same faulty secret key in memory, they will still match, fault remains undetected. An algorithm with such a countermeasure can provide fault tolerance against transient faults but not permanent faults. Hence, it is recommended to add spatial redundancy.

Spatial Redundancy. Spatial Redundancy involves simultaneous execution of N instances of a critical task for N level of redundancy and comparing their results for fault detection. If we store redundant copies of the original secret key during the key generation process at different memory locations, they can be used in parallel computations of signature generation using spatial redundancy technique. To bypass this approach, the attacker will need to fault the same exact bits at both memory

locations which will be much harder because every cell in the DRAM is not faulty. An important point here is that for deterministic version of Dilithium, this approach can work in a straightforward manner because the same nonce y is generated for the same message signed twice. However, for the randomized version, we will also need to store a copy of the nonce y for redundant computation. On the performance side, computation based on spatial redundancy will have significant overhead than the normal computation because of the increased complexity of the algorithm. Therefore, the level of spatial redundancy needed to detect faults should be taken into consideration.

Verify-after-Sign. If there is any bit flip in the secret key s_1 of Dilithium, it will generate a faulty signature which will not be verified by the verification algorithm. Hence, the verification can be used as a fault detection mechanism and if done on the signing side, the sender can easily detect the existence of the attacker. As compared to double signing, this approach is approximately three times faster as the verification algorithm takes less time as compared to the signing algorithm. There is still a possibility that the attacker also faults the verification in a way which results in a valid signature but to the best of our knowledge, there has not been such an algorithm developed so far for Dilithium. This approach may also fail if the comparison instruction is skipped using an instruction skip fault similar to [32]. Verify-after-sign can also detect instruction skip faults on signing step 13 and the rejection sampling step 15 in Algorithm 2 [32]. Rejection sampling step is critical because if it is bypassed without a mitigation in place, it can reveal information about the secret key [13].

8. Conclusion

We have proposed the Signature Correction attack targeting Dilithium a Round 3 finalist in the NIST PQC competition. The attack requires single bit-flips in the secret key vector, which we have implemented using Rowhammer targeting the constant-time AVX2 implementation of Dilithium. By analyzing the faulty signatures and exploiting redundancy in the secret key encoding, our attack successfully recovered 1,851 bits (out of 3,072 bits) of the secret key. This enabled us to reduce the post quantum security level to 81-bits (quantum) and 89-bits (classical) for both primal and dual lattice attacks. The attack is also applicable to other variants and security levels. We have demonstrated the first fault attack which works on randomized as well as deterministic versions of Dilithium. Our Signature Correction attack is independent of the fault mechanism but we have used Rowhammer to demonstrate the attack as it is a software only attack and does not need physical access. This can be very critical in case of cloud scenarios where the attacker can launch an attack remotely and leak secret information by using only faulty signatures. We have shown how few bits of secret key significantly reduce the security strength of Dilithium using the lattice attacks especially when the secret key encoding is exploited as in the analysis shown in this paper. Further, randomizing the nonce, a countermeasure commonly implemented in the design of signature schemes to thwart fault attacks, is not sufficient in practice as demonstrated by our attack.

Acknowledgment

We thank our anonymous reviewers for their insightful comments for improving the quality of this paper. This work is supported by U.S. Department of State, Bureau of Educational and Cultural Affairs' Fulbright Program and by the National Science Foundation under grants CNS-1814406, CNS-2026913 and CNS-1931639.

References

- [1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.
- [2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.
- [3] NIST, "Post-quantum cryptography standardization," <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization>, 2017.
- [4] J. Ding, J. Deaton, K. Schmidt, Vishakha, and Z. Zhang, "Cryptanalysis of the lifted unbalanced oil vinegar signature scheme," Cryptology ePrint Archive, Report 2019/1490, 2019, <https://eprint.iacr.org/2019/1490>.
- [5] P. Ravi, D. B. Roy, S. Bhasin, A. Chattopadhyay, and D. Mukhopadhyay, "Number "not used" once-practical fault attack on pqm4 implementations of NIST candidates," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2019, pp. 232–250.
- [6] D. Dachman-Soled, L. Ducas, H. Gong, and M. Rossi, "Lwe with side information: attacks and concrete security estimation," in *Annual International Cryptology Conference*. Springer, 2020, pp. 329–358.
- [7] A. Greuet, S. Montoya, and G. Renault, "Attack on LAC key exchange in misuse situation," in *International Conference on Cryptology and Network Security*. Springer, 2020, pp. 549–569.
- [8] D. Apon, R. Perlner, A. Robinson, and P. Santini, "Cryptanalysis of ledacrypt," in *Annual International Cryptology Conference*. Springer, 2020, pp. 389–418.
- [9] Y. Son, "A note on parameter choices of round5," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 949, 2019.
- [10] Y. Son and J. H. Cheon, "Revisiting the hybrid attack on sparse and ternary secret LWE," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1019, 2019.
- [11] R. Perlner and D. Smith-Tone, "Rainbow band separation is better than we thought," *Cryptology ePrint Archive*, 2020.
- [12] D. Kales and G. Zaverucha, "Forgery attacks on mqdssv2. 0," 2019.
- [13] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: A lattice-based digital signature scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 238–268, 2018.
- [14] D. Stebila and M. Mosca, "Post-quantum key exchange for the internet and the open quantum safe project," in *International Conference on Selected Areas in Cryptography*. Springer, 2016, pp. 14–37.
- [15] PQShield, "Think openly, build securely post-quantum standards ready," <https://pqshield.com>, 2021.
- [16] QuSecure, "Scalable cybersecurity for the post-quantum enterprise," <https://www.qusecure.com>, 2021.
- [17] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, D. Moody, R. Peralta *et al.*, "Status report on the second round of the NIST post-quantum cryptography standardization process," *US Department of Commerce, NIST*, 2020.
- [18] L. G. Bruinderink, A. Hülsing, T. Lange, and Y. Yarom, "Flush, gauss, and reload—a cache attack on the bliss lattice-based signature scheme," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2016, pp. 323–345.

- [19] 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.
- [20] 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.
- [21] P. Ravi, M. P. Jhanwar, J. Howe, A. Chattopadhyay, and S. Bhasin, "Side-channel assisted existential forgery attack on Dilithium-A NIST PQC candidate." *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 821, 2018.
- [22] P. Pessl and R. Primas, "More practical single-trace attacks on the number theoretic transform," in *International Conference on Cryptology and Information Security in Latin America*. Springer, 2019, pp. 130–149.
- [23] R. Primas, P. Pessl, and S. Mangard, "Single-trace side-channel attacks on masked lattice-based encryption," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 513–533.
- [24] T. Espitau, P.-A. Fouque, B. Gérard, and M. Tibouchi, "Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1857–1874.
- [25] J.-P. D'Anvers, M. Tiepelt, F. Vercauteren, and I. Verbauwhede, "Timing attacks on error correcting codes in post-quantum schemes," in *Proceedings of ACM Workshop on Theory of Implementation Security Workshop*, 2019, pp. 2–9.
- [26] A. Park, K.-A. Shim, N. Koo, and D.-G. Han, "Side-channel attacks on post-quantum signature schemes based on multivariate quadratic equations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 500–523, 2018.
- [27] A. Askeland and S. Rønjom, "A side-channel assisted attack on NTRU," *Cryptology ePrint Archive*, 2021.
- [28] E. Karabulut and A. Aysu, "Falcon down: Breaking falcon post-quantum signature scheme through side-channel attacks," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 691–696.
- [29] I.-J. Kim, T. Lee, J. Han, B.-Y. Sim, and D.-G. Han, "Novel single-trace ML profiling attacks on NIST 3 round candidate dilithium." *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1383, 2020.
- [30] K. Mus, S. Islam, and B. Sunar, "Quantumhammer: A practical hybrid attack on the LUOV signature scheme," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1071–1084.
- [31] L. G. Bruinderink and P. Pessl, "Differential fault attacks on deterministic lattice signatures," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 21–43, 2018.
- [32] P. Ravi, M. P. Jhanwar, J. Howe, A. Chattopadhyay, and S. Bhasin, "Exploiting determinism in lattice-based signatures: practical fault attacks on pqm4 implementations of NIST candidates," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019, pp. 427–440.
- [33] P. Pessl and L. Prokop, "Fault attacks on CCA-secure lattice KEMs," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 37–60, 2021.
- [34] J. Krämer and M. Loiero, "Fault attacks on UOV and RAINBOW," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2019, pp. 193–214.
- [35] N. Bindel, J. Buchmann, and J. Krämer, "Lattice-based signature schemes and their sensitivity to fault attacks," in *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2016, pp. 63–77.
- [36] T. Espitau, P.-A. Fouque, B. Gérard, and M. Tibouchi, "Loop-abort faults on lattice-based fiat-shamir and hash-and-sign signatures," in *International Conference on Selected Areas in Cryptography*. Springer, 2016, pp. 140–158.
- [37] N. Bindel, J. Kramer, and J. Schreiber, "Special session: hampering fault attacks against lattice-based signature schemes-countermeasures and their efficiency," in *2017 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2017, pp. 1–3.
- [38] A. Genêt, M. J. Kannwischer, H. Pelletier, and A. McLaughlan, "Practical fault injection attacks on SPHINCS," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 674, 2018.
- [39] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One bit flips, one cloud flops: Cross-VM row hammer attacks and privilege escalation," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 19–35.
- [40] L. Cojocar, J. Kim, M. Patel, L. Tsai, S. Saroiu, A. Wolman, and O. Mutlu, "Are we susceptible to rowhammer? an end-to-end methodology for cloud providers," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 712–728.
- [41] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 353–367.
- [42] S. Bai and S. D. Galbraith, "An improved compression technique for signatures based on learning with errors," in *Cryptographers' Track at the RSA Conference*. Springer, 2014, pp. 28–47.
- [43] V. Lyubashevsky, "Fiat-shamir with aborts: Applications to lattice and factoring-based signatures," in *Advances in Cryptology – ASIACRYPT 2009*, M. Matsui, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 598–616.
- [44] 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)*. Austin, TX: USENIX Association, Aug. 2016, pp. 565–581.
- [45] 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," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.
- [46] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, p. 71, 2015.
- [47] V. Van Der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic rowhammer attacks on mobile platforms," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 1675–1689.
- [48] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoecl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 245–261.
- [49] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in javascript," in *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 2016, pp. 300–321.
- [50] F. de Ridder, P. Frigo, E. Vannacci, H. Bos, C. Giuffrida, and K. Razavi, "SMASH: Synchronized many-sided rowhammer attacks from JavaScript," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1001–1018.
- [51] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, "Throwhammer: Rowhammer attacks over the network and defenses," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, Jul. 2018, pp. 213–226.
- [52] 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.
- [53] Z. Weissman, T. Tiemann, D. Moghimi, E. Custodio, T. Eisenbarth, and B. Sunar, "Jackhammer: Efficient rowhammer on heterogeneous fpga-cpu platforms," *arXiv preprint arXiv:1912.11523*, 2019.

- [54] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, “RAMBleed: Reading bits in memory without accessing them,” in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 695–711.
- [55] G. Irazoqui, T. Eisenbarth, and B. Sunar, “MASCAT: Stopping microarchitectural attacks before execution.” *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 1196, 2016.
- [56] M. Chiappetta, E. Savas, and C. Yilmaz, “Real time detection of cache-based side-channel attacks using hardware performance counters,” *Applied Soft Computing*, vol. 49, pp. 1162–1174, 2016.
- [57] T. Zhang, Y. Zhang, and R. B. Lee, “Cloudradar: A real-time side-channel attack detection system in clouds,” in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2016, pp. 118–140.
- [58] N. Herath and A. Fogh, “These are not your grand Daddys cpu performance counters—cpu hardware performance counters for security,” *Black Hat Briefings*, 2015.
- [59] M. Payer, “HexPADS: a platform to detect “stealth” attacks,” in *International Symposium on Engineering Secure Software and Systems*. Springer, 2016, pp. 138–154.
- [60] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, “Flush+ Flush: a fast and stealthy cache attack,” in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 279–299.
- [61] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, “ANVIL: Software-based protection against next-generation rowhammer attacks,” *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 743–755, 2016.
- [62] J. Corbet, “Defending against Rowhammer in the kernel,” Oct. 2016, <https://lwn.net/Articles/704920/>.
- [63] 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)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 117–130.
- [64] J. S. S. T. Association, *Low Power Double Data Rate 4 (LPDDR4)*, Jan. 2020, <https://www.jedec.org/standards-documents/docs/jesd209-4b>.
- [65] D.-H. Kim, P. J. Nair, and M. K. Qureshi, “Architectural support for mitigating row hammering in DRAM memories,” *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 9–12, 2014.
- [66] M. Ghasempour, M. Lujan, and J. Garside, “Armor: A run-time memory hot-row detector,” 2015.
- [67] IBM, “IBM Chipkill Memory: Advanced ECC memory for the IBM Netfinity 7000 M10,” 2019, http://ps-2.kev009.com/pccbbs/pc_servers/chipkillf.pdf.
- [68] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, “Exploiting correcting codes: On the effectiveness of ECC memory against rowhammer attacks,” in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 55–71.
- [69] P. Frigo, E. Vannacc, H. Hassan, V. Van Der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, “TRRepass: Exploiting the many sides of target row refresh,” in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 747–762.
- [70] 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.
- [71] L. Xu, R. Yu, L. Wang, and W. Liu, “Memway: in-memory waylaying acceleration for practical rowhammer attacks against binaries,” *Tsinghua Science and Technology*, vol. 24, no. 5, pp. 535–545, 2019.
- [72] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, “Post-quantum key Exchange—A New Hope,” in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 327–343.
- [73] Y. Chen and P. Q. Nguyen, “BKZ 2.0: Better lattice security estimates,” in *Advances in Cryptology – ASIACRYPT 2011*, D. H. Lee and X. Wang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–20.
- [74] C. P. Schnorr and M. Euchner, “Lattice basis reduction: Improved practical algorithms and solving subset sum problems,” *Math. Program.*, vol. 66, no. 2, p. 181–199, Sep. 1994.
- [75] A. Becker, L. Ducas, N. Gama, and T. Laarhoven, “New directions in nearest neighbor searching with applications to lattice sieving,” in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA ’16. USA: Society for Industrial and Applied Mathematics, 2016, p. 10–24.
- [76] T. Laarhoven, “Sieving for shortest vectors in lattices using angular locality-sensitive hashing,” in *Advances in Cryptology – CRYPTO 2015*, R. Gennaro and M. Robshaw, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 3–22.
- [77] —, “Search problems in cryptography: from fingerprinting to lattice sieving,” Ph.D. dissertation, Mathematics and Computer Science, Feb. 2016, proefschrift.
- [78] T. Laarhoven, M. Mosca, and J. Pol, “Finding shortest lattice vectors faster using quantum search,” *Des. Codes Cryptography*, vol. 77, no. 2–3, p. 375–400, Dec. 2015.
- [79] J. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila, “Frodo: Take off the ring! practical, quantum-secure key exchange from LWE,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1006–1018.
- [80] A. Casanova, J.-C. Faugere, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem, “Gemss: a great multivariate short signature,” Ph.D. dissertation, UPMC-Paris 6 Sorbonne Universités; INRIA Paris Research Centre, MAMBA Team ..., 2017.
- [81] J. Ding and D. Schmidt, “Rainbow, a new multivariable polynomial signature scheme,” in *International conference on applied cryptography and network security*. Springer, 2005, pp. 164–175.
- [82] J. Sepulveda, A. Zankl, and O. Mischke, “Cache attacks and countermeasures for NTRUEncrypt on MPSoCs: Post-quantum resistance for the IOT,” in *2017 30th IEEE International System-on-Chip Conference (SOCC)*, 2017, pp. 120–125.
- [83] J. H. Silverman and W. Whyte, “Timing attacks on NTRUEncrypt via variation in the number of hash calls,” in *Topics in Cryptology – CT-RSA 2007*, M. Abe, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 208–224.
- [84] J.-P. D’Anvers, M. Tiepelt, F. Vercauteren, and I. Verbauwhede, “Timing attacks on error correcting codes in post-quantum schemes,” in *Proceedings of ACM Workshop on Theory of Implementation Security Workshop*, ser. TIS’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2–9.
- [85] N. Bindel, J. Buchmann, J. Krämer, H. Mantel, J. Schickel, and A. Weber, “Bounding the cache-side-channel leakage of lattice-based signature schemes using program semantics,” in *International Symposium on Foundations and Practice of Security*. Springer, 2017, pp. 225–241.
- [86] L. Castelnovi, A. Martinelli, and T. Prest, “Grafting Trees: a fault attack against the SPHINCS framework,” in *International Conference on Post-Quantum Cryptography*. Springer, 2018, pp. 165–184.
- [87] K. Xagawa, A. Ito, R. Ueno, J. Takahashi, and N. Homma, “Fault-injection attacks against NIST’s post-quantum cryptography round 3 kem candidates,” in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2021, pp. 33–61.
- [88] F. Valencia, T. Oder, T. Güneysu, and F. Regazzoni, “Exploring the vulnerability of R-LWE encryption to fault attacks,” in *Proceedings of the Fifth Workshop on Cryptography and Security in Computing Systems*, ser. CS2 ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 7–12.
- [89] A. A. Kamal and A. M. Youssef, “Fault analysis of the NTRUSign digital signature scheme,” *Cryptography Commun.*, vol. 4, no. 2, p. 131–144, Jun. 2012.

- [90] A. A. Kamal and A. Youssef, "Strengthening hardware implementations of NTRUEncrypt against fault analysis attacks," *Journal of Cryptographic Engineering*, vol. 3, pp. 227–240, 2013.
- [91] B. Gulmezoglu, A. Moghimi, T. Eisenbarth, and B. Sunar, "Fortuneteller: Predicting microarchitectural attacks via unsupervised deep learning," *arXiv preprint arXiv:1907.03651*, 2019.
- [92] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1555–1571, 2019.
- [93] J. Standard, "Double Data Rate (DDR) SDRAM specification," *JEDEC Solid State Technology Assoc*, 2005.
- [94] K. K.-W. Chang, D. Lee, Z. Chishti, A. R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving DRAM performance by parallelizing refreshes with accesses," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014, pp. 356–367.
- [95] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-aware intelligent DRAM refresh," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 1–12, 2012.