When Frodo Flips: End-to-End Key Recovery on FrodoKEM via Rowhammer

Michael Fahr Jr.
University of Arkansas
Fayetteville, AR, USA
mjfahr@uark.edu

Hunter Kippen
University of Maryland
College Park, MD, USA
hkippen@umd.edu

Andrew Kwong
University of Michigan
Ann Arbor, MI, USA
ankwong@umich.edu

Thinh Dang
George Washington University, NIST
Washington, D.C., USA
Gaithersburg, MD, USA
thinh@gwu.edu,thinh.dang@nist.gov

Jacob Lichtinger
NIST
Gaithersburg, MD, USA
jacob.lichtinger@nist.gov

Dana Dachman-Soled
University of Maryland
College Park, MD, USA
danadach@ece.umd.edu

Daniel Genkin
Georgia Institute of Technology
Atlanta, GA, USA
genkin@gatech.edu

Alexander Nelson
University of Arkansas
Fayetteville, AR, USA
ahnelson@uark.edu

Ray Perlner
NIST
Gaithersburg, MD, USA
ray.perlner@nist.gov

Arkady Yerukhimovich
George Washington University
Washington, D.C., USA
arkady@gwu.edu

Daniel Apon
MITRE
McLean, VA, USA
dapon@mitre.org

ABSTRACT

In this work, we recover the private key material of the FrodoKEM key exchange mechanism as submitted to the NIST Post Quantum Cryptography (PQC) standardization process. The new mechanism that allows for this is a Rowhammer-assisted *poisoning* of the FrodoKEM Key Generation (KeyGen) process. The Rowhammer side-channel is a hardware-based security exploit that allows flipping bits in DRAM by "hammering" rows of memory adjacent to some target-victim memory location by repeated memory accesses. Using Rowhammer, we induce the FrodoKEM software to output a higher-error Public Key (PK), $(\mathbf{A}, \mathbf{B} = \mathbf{AS} + \widetilde{\mathbf{E}})$, where the error $\widetilde{\mathbf{E}}$ is modified by Rowhammer.

Then, we perform a decryption failure attack, using a variety of publicly-accessible supercomputing resources running on the order of only 200,000 core-hours. We delicately attenuate the decryption failure rate to ensure that the adversary's attack succeeds practically, but so honest users cannot easily detect the manipulation.

Achieving this public key "poisoning" requires an extreme engineering effort, as FrodoKEM's KeyGen runs on the order of 8 *milli*seconds. (Prior Rowhammer-assisted attacks against cryptography require as long as 8 hours of persistent access.) In order to handle this real-world timing condition, we require a wide variety of prior and brand new, low-level engineering techniques, including e.g. memory massaging algorithms – i.e. "Feng Shui" – and a precisely-targeted performance degradation attack on the extendable output function SHAKE.

We explore the applicability of our techniques to other lattice-based KEMs in the NIST PQC Round 3 candidate-pool, e.g. Kyber, Saber, etc, as well as the difficulties that arise in the various settings. To conclude, we discuss various simple countermeasures to protect implementations against this, and similar, attacks.

1 INTRODUCTION

In recent years, the possible emergence of a cryptographically relevant quantum computer (CRQC), a device that exploits quantum-mechanical phenomena to break cryptographic systems, has become more of a reality. A substantial amount of research has gone into the construction of such machines, resulting in increasingly larger quantum computers. For example, in 2019, researchers at Google announced an experimental realization of "quantum supremacy" [5], the demonstration of a programmable quantum device solving a problem that is infeasible for any conventional computer. If fully realized, a CRQC would be capable of undermining the security of digital communications on the Internet and elsewhere [1].

The only practical counter to this threat is the development and deployment of quantum-resistant (or post-quantum) cryptography. In 2016, the U.S. National Institute of Standards and Technology (NIST) announced the beginning of its Post-Quantum Cryptography (PQC) standardization process [46] aimed to standardize quantum-resistant public-key cryptographic algorithms. The conclusion of Round 3 of this project is imminent, resulting in NIST announcing its decisions for the first post-quantum public-key encryption / key establishment mechanism (KEM) and digital signature standards. The pool of candidate-algorithms in the NIST process has been reduced from a large field of 69 submissions in 2017 to a small set of Round 3 finalists and alternates. Among these remaining candidates, lattice-based cryptography plays an especially prominent role, as 5 of the 7 finalists are lattice-based.

Among the lattice-based constructions considered in the 3rd Round, the FrodoKEM encryption protocol has the most mathematically conservative, security-conscious design. It is the only lattice candidate that has no special algebraic structure but instead bases its security on the *plain* Learning With Errors (LWE) problem [55]. In particular, FrodoKEM was recommended by the German Federal Office for Information Security (BSI) [25] in 2020 as "suitable for long-term confidentiality protection."

Next, in addition to security against cryptanalysis, NIST has also made clear throughout the standardization process that PQC candidates also should be resilient against side-channel attacks [9, 31, 48, 52, 53, 59, 64, 67]. While some side-channel attack vectors can be defended against using constant-time coding techniques, other actively-induced effects, such as Rowhammer induced bit flips, cannot be as easily mitigated through careful coding practices. Relatively little is known about the resistance of current PQC constructions to active side-channels like Rowhammer. There are only two prior works investigating such attacks against NIST PQC algorithms, and they examined only the case of digital signatures: specifically, the LUOV and Dilithium signature schemes [32, 42]. In this work, we embark on the task of investigating the Rowhammer resilience of NIST's post-quantum KEM candidates, focusing foremost on FrodoKEM as a representative "hard target."

Specifically, we ask the following main questions:

How resilient are PQC KEM constructions to Rowhammer attacks? What would it take for an adversary to mount such attacks, and what information can be extracted using them?

1.1 Our Contributions

We demonstrate the first end-to-end implementation of a successful key recovery attack against FrodoKEM using Rowhammer. At a high level, our attack works as follows. First, we use Rowhammer to poison FrodoKEM's KeyGen process. Then, we use a supercomputer to extract private-key material. Next, we synthesize this data using a tailored key-recovery algorithm. Finally, any individual FrodoKEM-encrypted session-key can be recovered in around 2 minutes on a commodity laptop.

Our work demonstrates the near-term and high importance of protecting lattice-based cryptography's key generation processes from active side-channel attacks. Especially, we highlight the scenario of running a lattice KEM's KeyGen in a cloud computing environment, where an adversary will have access to a common, shared-memory architecture on which honest users run KeyGen: In such a setting, honest users are particularly vulnerable to our line of attack and should aim to protect themselves with intention.

1.2 Overview of Our New Attack

The main ideas underlying our attack on FrodoKEM are as follows.

Decryption Failure Attacks. We begin with an observation made previously [35] that many lattice-based KEMs, including FrodoKEM, have a non-zero decryption failure rate (DFR),[1] and this may lead to a security vulnerability. That is, validly encrypted ciphertexts may occasionally fail to decrypt properly, and moreover, such failing ciphertexts reveal information about the secret key used in decryption. This has led to multiple *decryption failure attacks* and related attack variants in the literature, beginning with Fluhrer's attack on Ring-LWE [21] with many improvements later (e.g. [7, 19, 20, 51]) targeting CPA-secure schemes. These attacks typically make adaptive decryption queries to the receiver of a KEM using carefully crafted ciphertexts. They rely on information of whether a failure occurred or not to gradually recover the secret key.

Protecting FrodoKEM Against Decryption Failures. Decryption failure attacks are mitigated in FrodoKEM and other NIST PQC Round 3 KEMs by use of a Fujisaki-Okamoto (FO) transform [24, 30, 57] that converts a base (IND- or OW-) CPA-secure encryption scheme into a CCA-secure KEM. At a high level, an FO-like transform samples the randomness used to construct a given ciphertext by applying a hash function modeled as a random oracle to its plaintext message. After decryption of a candidate-message, the receiver (deterministically) recomputes a ciphertext encrypting that message and checks if this rederived ciphertext exactly matches the initial ciphertext it received. This makes it impossible for an adversary to craft arbitrary ciphertexts (or maul honestly-constructed ciphertexts), as an overwhelming fraction of these will fail the FO re-encryption check.

Therefore, an adversary against such FO-transformed CCA-secure KEMs is forced to run the honest encryption procedure to produce ciphertexts that will not be rejected outright. Indeed, the parameters of KEM candidates in the 3rd round of the NIST PQC process are chosen to balance the work of an adversary who attempts a decryption failure type of attack and an adversary who attempts a direct cryptanalytic attack on the mathematics of the system.[2]

Failure-Boosting Attacks. Further works [8, 16, 17] have explored failure-boosting attacks against the CCA-secure forms of NIST PQC candidate-KEMs. The idea here is that the receiver in a real-world KEM will perform only so many decryptions on behalf of various senders attempting to establish a common session key. (The NIST PQC Call For Proposals [44] sets an absolute upper limit of 2^{64} decryptions.) If a random ciphertext fails to decrypt with probability (say) 2^{-128} or 2^{-256}, it's clearly highly unlikely that even 2^{64} decryption-query attempts will yield a single failing ciphertext. But an adversary can do better by querying the random oracle locally, and generating a large list of candidate-ciphertexts to query for decryption. For example, lattice-based ciphertexts with a higher-norm or heavier weight in certain integer-coordinates are more likely to fail to decrypt than a random ciphertext. A failure-boosting adversary chooses a subset of the most-likely-to-fail ciphertexts, and queries only on those. While the above description summarizes the the current state-of-the-art for decryption failure attacks, modern analyses show that the carefully-set parameters of NIST Round 3 KEMs prevent such lines of attack from succeeding in practice.

Intuition for Our Attack. This is where our new Rowhammer-assisted attack comes in. The Rowhammer side-channel is a hardware-based security exploit that allows flipping bits in DRAM by "hammering" rows of memory adjacent to some target-victim memory location by repeated memory accesses. Specifically,

[1] This is not an inherent requirement. Our attack may also succeed against rigid lattice-based KEMs with perfect correctness. We defer the details to the body of the paper.

[2] As an interesting historical note, CRYSTALS-Kyber-512 changed a certain binomial-sampling parameter from 2 to 3 between the 2nd and 3rd Rounds of the NIST PQC process explicitly to better balance the cost of these attacks.

physically-adjacent memory cells interact electrically between themselves, and when a sufficient charge builds up in a given capacitor, it may discharge into adjacent capacitors (where the adversary does not have read/write permissions), causing their logical bits to flip.

By targeting the locations in RAM where the LWE secret key and error are stored during the key generation procedure, we can use Rowhammer to flip some of the higher-order bits of the LWE secret key material. This, in turn, means that the magnitude of certain secret and error coordinates are far higher than would be naturally generated by an honest run of the key generation algorithm. The effect is that decryption failures become somewhat more likely (although not so much that honest users will perceive the difference) – but for an adversary who knows precisely which bits were flipped, obtaining decryption failures via the failure-boosting approach becomes *extremely* more likely (e.g. 2^{-128} becomes $\approx 2^{-12}$).

While this constitutes the core intuition for our new attack, a plethora of additional issues need to be considered to make this approach feasible in the real world.

Rowhammer Timing. A first complication is that Rowhammer bit-flipping requires a minimum window of time at least as long as the electrical refresh rate of the DRAM device being targeted. On typical devices, this is 64 milliseconds, yet FrodoKEM (the slowest of the NIST lattice-based KEM candidates) runs in around 8 milliseconds. We note that FrodoKEM and other NIST PQC KEMs also compute a hash of the generated public key and publish this hash with the associated algebraic material. In turn, this requires that Rowhammer manipulation completes very shortly after (or before!) the natural KeyGen computation completes. In other words, the algebraic computation in the key generation process inherently must last for at least 64 milliseconds for Rowhammer to work. To overcome this challenge, we rely on an additional performance degradation side-channel (to sufficiently delay the execution of the FrodoKEM key generation procedure), so that our Rowhammer attack can do its work.

Memory Profiling. But this step is not enough. A second concern is that the Rowhammer is sensitive to the physical characteristics of individual DRAM modules in the victim device, decided arbitrarily due to process variation during manufacturing at a foundry. For any given page of RAM, some bits will flip frequently when subjected to hammering, and other bits will flip only very slowly. The solution is to profile the target device (before the Lattice KEM algorithm comes online), by hammering every position in the RAM up front, to determine which bits on which pages are more likely to flip.

Memory Massaging. Given such knowledge of particularly "flippy" locations in various pages of the victim DRAM device, we now want to ensure that the FrodoKEM algorithm allocates its memory in a specific manner – exactly aligning with the precise bits of the FrodoKEM algebraic material that we want to manipulate. The mechanism for doing so is "Feng Shui," or memory massaging. Here, we force the victim to allocate its memory in the exact pages of DRAM that we want, by exploiting the low-level details of the data structure underlying the allocation of Linux page frame caches.

Side Stepping Memory Bit Masks. Yet even this is not quite enough. Our Rowhammer procedure physically flips bits in a one-sided manner: from 0 to 1. There are two problems that can arise with this construction. First, logical bits stored in DRAM are distinct from the physical bits stored in DRAM. In particular, every time the device is booted up, a random bit-mask is applied (akin to a one-time pad). This pad will be consistent, but for N possible pads, the attack succeeds with $1/N$ probability if the machine is reset between profiling and the attack. The solution is to re-profile after any restarts to ensure that the mask does not affect bit locations. Second, 2's complement signed representation dramatically reduces the success probability. For each column, we want to place a single 256-bit flip. The value prior to the flip is equally probable to be negative or positive. The consequence of this representation is that a negative value being targeted to induce a 256-bit flip from 0 to 1 will be unsuccessful because the bit is already a 1. This means that each column can be attacked with probability 2^{-1}, and that each of these locations is an independent probability. So our attack succeeds with additional probability 2^{-N} where N is the number of columns that need to be poisoned, depending on the random sampling of E in the key-generation process.

Generating Failing Ciphertexts. Now we have properly poisoned a FrodoKEM public key. Once the key material has been poisoned, we next need to find sufficiently many failing ciphertexts. For this purpose, we use supercomputing resources (described in Section 5) to run an honest encryption procedure on over a trillion distinct messages. Rather than requesting decryptions on this many messages – which would be outside the bounds of an honest receiver's acceptable workload, and thus certainly detected and rejected by a server – we use our knowledge of the hammered positions to filter out ciphertexts that are unlikely to cause a decryption failure. Specifically, we only keep ciphertexts that have either a large positive or a large negative value in the targeted bit-locations, thus maximizing the probability of failure. By asking that only such ciphertexts be decrypted, we also significantly reduce the probability of detection.

Key Recovery. To perform key recovery given the failing ciphertexts, we present and analyze the following simple algorithm: (1) Construct a set of vectors from the failing ciphertexts generated in the previous stage, (2) Scale these vectors up by a constant factor that depends on the FrodoKEM key distribution and on the rowhammered locations, (3) Take the component-wise average of these scaled-up vectors, (4) Round the resulting vector component-wise to the nearest integer and output this as the candidate key. Comparing our approach with the lattice reduction approach of [15], we find that incorporating the same number of failing ciphertexts using the Toolkit from [15] yields an SVP instance with an estimated hardness of 150 bikz (corresponding to a bit-security of approximately 40), which would not be solvable on a commodity laptop. Further, even obtaining this 150-bikz SVP instance by incorporating decryption failure information using the Toolkit of [15] (as opposed to just computing the hardness *estimates* referenced above), was too computationally intensive for us to run on a commodity laptop due to the large number of failing ciphertexts required in our setting (the required large number of failing ciphertexts is due to the fact that the decryption failure threshold is effectively lowered in our attack, making failures more common, and resulting in less information gained about the secret key from each failure). Thus, for the current setting, our key recovery algorithm outlined above is far more computationally efficient than the approach of [15].

Attacking Session Keys. In our experiment, the attack described thus far allowed us to recover 7/8 columns of the master secret key. However, it is actually computationally hard to recover the remaining bits of the master key. So, instead, we turn to recover the session keys. We show a simple procedure that, when most of the master secret key is known, can recover the full session key by simply trying all possible values for the remaining bits of the session key. We show that this incremental session key recovery step can complete in only a couple minutes on a commodity laptop.

1.3 Attacker Model

Our attack relies on inducing bit flips during FrodoKEM's key generation process. We assume the common threat model for Rowhammer attacks. In this model, the attacker and FrodoKEM's key generation process run on the same physical hardware. This can occur with mutually distrusting virtual machines running on the same server, or when the attacker and victim run in two separate processes under the same OS. This threat model is consistent with the current body of literature on Rowhammer [68], as well as the majority of microarchitectural attack papers [69].

Finally, given that FrodoKEM is a CCA-secure scheme, we also assume that the attacker can induce the victim to decrypt honestly generated ciphertexts of the attacker's choice.

1.4 Artifacts and Current Status

Artifacts. For completeness, we provide our artifacts at this URL (See `Readme` first): https://github.com/a-as-plus-e/FrodoFLIP

Current Status. After acceptance of this paper, NIST selected Kyber as the initial post-quantum KEM [2]. Nonetheless, Frodo remains a post-quantum recommendation of Germany's BSI [25] and the core, mathematical foundation underlying Kyber [47].

1.5 Paper Organization

The rest of this paper is organized as follows. First, in Section 2, we review some necessary background and prior work as well as notation and definitions. Then, in Sections 3 - 6, we dive into the details of each component of our attack. Specifically, in Section 3, we describe how decryption failures can be used to recover a FrodoKEM secret key. Then, in Section 4, we describe how to use a Rowhammer attack to enable creation of decryption failures in FrodoKEM. In Section 5 we describe how to use a supercomputer to find sufficiently many failing ciphertexts for our attack. Finally, in Section 6, we conclude our attack by showing how we can recover a FrodoKEM session-key. We then discuss how our techniques can be extended to other lattice-based KEMs in Section 7 and possible countermeasures in Section 8.

2 BACKGROUND

We now provide background on the FrodoKEM protocol as well as on the Rowhammer attack that we use to break its security.

2.1 Notation

We use bold lower case letters to denote vectors, and bold upper case letters to denote matrices. We use column notation for vectors, and start indexing from 0. We denote by $\mathbf{I}_n$ the n-dimensional identity matrix and denote by $\mathbf{x} \cdot \mathbf{y}$ the inner product of vectors $\mathbf{x}, \mathbf{y}$ of the same dimension. In addition, we make use of Einstein notation to delineate between rows and columns of a matrix (e.g. For a matrix $\mathbf{A}$, $\mathbf{a}_i$ is the i^{th} row of the matrix, and $\mathbf{a}^j$ is the j^{th} column of the matrix). We denote by $(\mathbf{x}||\mathbf{y})$ the concatenation of two column vectors $\mathbf{x}, \mathbf{y}$, which is a column vector whose dimension is the sum of the dimensions of $\mathbf{x}$ and $\mathbf{y}$. Random variables—i.e. variables whose values depend on outcomes of a random experiment—are denoted with lowercase calligraphic letters e.g. $\mathcal{a}, \mathcal{b}, \mathcal{e}$, while random vectors are denoted with uppercase calligraphic letters e.g. $\mathcal{C}, \mathcal{X}, \mathcal{Z}$.

2.2 Statistics

Definition 1 (Univariate normal distribution)**.** We denote by $\mathcal{N}(\mu, \sigma^2)$ the univariate normal (Gaussian) distribution with mean μ, variance σ^2, and probability density function (pdf)

$$x \mapsto \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right).$$

Definition 2 (Multivariate normal distribution)**.** Let $d \in \mathbb{Z}$, $\boldsymbol{\mu} \in \mathbb{Z}^d$ and let Σ be a positive definite matrix of dimension $d \times d$. We denote by $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ the multivariate normal (Gaussian) distribution with mean $\boldsymbol{\mu}$, covariance Σ, and probability density function (pdf)

$$\mathbf{x} \mapsto \frac{1}{\sqrt{(2\pi)^d \cdot \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})^T\right).$$

Definition 3. The error function, denoted `erf` is defined as:

$$\texttt{erf}(z) := \frac{2}{\sqrt{\pi}} \int_0^z \exp(-t^2)\, dt.$$

$\texttt{erf}(z)$ is the probability that an x sampled from $\mathcal{N}(0,1)$ falls in the range $[-z, z]$.

2.3 FrodoKEM

Definition 4 (Public-key encryption scheme (PKE))**.** A public-key encryption scheme is a tuple of algorithms (KeyGen, Enc, Dec):

- KeyGen() outputs (pk, sk), where pk is the public key and sk is the secret key.
- Enc(pk, μ) takes as input a public key pk and a message μ and outputs a ciphertext c.
- Dec(sk, c) takes as input a secret key sk and a ciphertext c and outputs a message μ, or fails.

A public-key encryption scheme is correct if an honest in-order execution of KeyGen, Enc, and Dec, results in Dec outputting the same message that is input into Enc (with overwhelming probability). The relevant notion of security for a public-key encryption scheme in our discussion is IND-CPA. A public-key encryption scheme is IND-CPA if any adversary cannot distinguish between ciphertexts of two adversarially-chosen messages with non-negligible advantage.

Definition 5 (Key Establishment Mechanism (KEM))**.** A key establishment mechanism (KEM) is a tuple of algorithms (KeyGen, Enc, Dec):

- KeyGen() outputs (pk, sk), where pk is the public key and sk is the secret key.
- Enc(pk) takes as input a public key pk and outputs (c, k), where c is the ciphertext that encapsulates the shared session key k.

- $\mathrm{Dec}(sk, c)$ takes as input a secret key sk and a ciphertext c, and outputs a shared session key k, or fails.

A key-encapsulation mechanism is correct if an honest in-order execution of KeyGen, Enc, and Dec results in the same shared session key output by Enc and Dec (with overwhelming probability). Moreover, a KEM is IND-CCA if any adversary with access to a decapsulation oracle, on input of a random challenge session key and a challenge ciphertext, cannot distinguish between the case when the session key is encapsulated in the ciphertext and the case when the session key is independent and uniformly random, with non-negligible advantage.

FrodoKEM is an IND-CCA key-encapusulation mechanism. FrodoKEM is constructed as a Fujisaki-Okamoto transform of FrodoPKE, an IND-CPA public-key encryption scheme whose security is based on the hardness of Learning with Errors. Learning with Errors was first defined and studied as the basis for a public-key cryptosystem by Regev in [55]. Lindner and Peikert later in [38] showed a more efficient public-key encryption scheme based on the hardness of Learning with Errors. FrodoPKE is an instantiation of Lindner and Peikert's construction. We give the definitions of Learning with Errors, both the search and decision variants, below. We note that the definitions we state here are of so-called normal-form Learning with Errors introduced in [41], which shows that normal-form Learning with Errors is at least as hard as Learning with Errors as originally defined by Regev in [55].

Definition 6 (LWE distribution)**.** The LWE distribution is parametrized by positive integers (m, n, q) and an error distribution χ over $\mathbb{Z}$. The LWE distribution is sampled by sampling a uniformly random $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}$, $\mathbf{s} \leftarrow \chi^n$, $\mathbf{e} \leftarrow \chi^m$, and outputting $(\mathbf{A}, \mathbf{b} := \mathbf{As} + \mathbf{e} \bmod q)$.

Definition 7 (The Search-LWE Problem)**.** Given $(\mathbf{A}, \mathbf{b})$ drawn from the LWE distribution, search-LWE problem asks to find $\mathbf{s}$ in the support of χ^n such that $\mathbf{e} := \mathbf{b} - \mathbf{As}$ is in the support of χ^m modulo q.

Definition 8 (The Decision-LWE Problem)**.** The decision-LWE problem asks to distinguish between the LWE distribution and the uniform distribution over $(\mathbb{Z}_q^{m \times n}, \mathbb{Z}_q^m)$.

It's been shown (e.g. in [55]) that there is a reduction from search-LWE to decision-LWE.

We give a high-level description of FrodoPKE in algorithms 1, 2, and 3, where we omit details about (pseudo)random generation using symmetric primitives, sampling from the error distribution χ, and bit-level representations. We point readers to the Frodo specification [3] for details. In the following, $n, \bar{m}, \bar{n}, B$, and q are integer parameters. Specifically, for Frodo640, $n = 640, \bar{m} = \bar{n} = 8, B = 2$, and $q = 32768$.

We now describe the *Encode* and *Decode* functions to complete our description of FrodoPKE. The *Encode* function encodes an integer k such that $0 \le k < 2^B \le q$ as an element in $\mathbb{Z}_q$:

$$Encode(k) := k \cdot q/2^B.$$

In Frodo, q is a power of 2, and therefore $q/2^B$ is an integer. The *Decode* function extracts a B-bit integer from an element of $\mathbb{Z}_q$:

$$Decode(k) := \lfloor c \cdot 2^B/q \rceil \bmod 2^B.$$

Algorithm 1 FrodoPKE.Keygen()

1: $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times n}$, $\mathbf{S} \leftarrow \chi^{n \times \bar{n}}$, $\mathbf{E} \leftarrow \chi^{n \times \bar{n}}$
2: $\mathbf{B} = \mathbf{AS} + \mathbf{E} \bmod q$
3: **return** $(pk = (\mathbf{A}, \mathbf{B}), sk = \mathbf{S})$

Algorithm 2 FrodoPKE.Enc(pk, μ)

1: $\mathbf{A}, \mathbf{B} = pk$
2: $\mathbf{S}', \mathbf{E}' \leftarrow \chi^{\bar{m} \times n}$
3: $\mathbf{E}'' \leftarrow \chi^{\bar{m} \times \bar{n}}$
4: $\mathbf{B}' = \mathbf{S}'\mathbf{A} + \mathbf{E}'$
5: $\mathbf{V} = \mathbf{S}'\mathbf{B} + \mathbf{E}''$
6: $(\mathbf{C}_1, \mathbf{C}_2) = (\mathbf{B}', \mathbf{V} + Encode(\mu))$
7: **return** $c = (\mathbf{C}_1, \mathbf{C}_2)$

Algorithm 3 FrodoPKE.Dec(sk, μ)

1: $\mathbf{S} = sk$
2: $(\mathbf{C}_1, \mathbf{C}_2) = c$
3: $\mathbf{M} = \mathbf{C}_2 - \mathbf{C}_1\mathbf{S}$
4: **return** $\mu = Decode(\mathbf{M})$

We extend the domain of *Encode* and *Decode* to vectors and matrices by entry-wise application. The *Encode* and *Decode* functions have an error-correcting property stated in [3, Lemma 2.18]. We re-state the result below.

Lemma 1. Let $q = 2^D$, $B \le D$. Then $Decode(Encode(k) + e) = k$ for any $k, e \in \mathbb{Z}$ such that $0 \le k < 2^B$ and $-q/2^{B+1} \le e < q/2^{B+1}$.

The correctness of FrodoPKE follows because the decryption computes

$$\begin{aligned}\mathbf{M} &= \mathbf{C}_2 - \mathbf{C}_1\mathbf{S} \\ &= Encode(\mu) + \mathbf{S}'\mathbf{E} - \mathbf{E}'\mathbf{S} + \mathbf{E}''.\end{aligned}$$

Let $\mathbf{E}''' = \mathbf{S}'\mathbf{E} - \mathbf{E}'\mathbf{S} + \mathbf{E}''$. By Lemma 1, if the entries of $\mathbf{E}'''$ are sufficiently small, we have that $Decode(\mathbf{M}) = \mu$. A lower bound on the probability of this event can be computed explicitly using Frodo's parameter search script in Frodo submission available at [45]. Conversely, if an entry in the decoding error $\mathbf{E}'''$ exceeds the decoding threshold $[-q/2^{B+1}, q/2^{B+1})$, a decoding failure will occur in the corresponding entry of μ.

Given an IND-CPA public-key encryption scheme such as FrodoPKE above, a generic Fujisaki-Okamoto transform can be applied to obtain an IND-CCA key-encapsulation mechanism. We give a high-level description of the Fujisaki-Okamoto transform used by FrodoKEM in algorithms 4, 5, and 6, where H is a cryptographic hash function (e.g. SHAKE) modeled as a random oracle.

For our attack, we consider Frodo640, the NIST level 1 (as hard as brute-force search on AES-128) parameter set of FrodoKEM. For Frodo640, the parameters are as follows: $n = 640, \bar{n} = \bar{m} = 8, q = 2^{15}, B = 2$. Therefore, the shared session key is decoded from an 8×8 matrix and the decoding threshold is $[-4096, 4096)$.

Algorithm 4 FrodoKEM.KeyGen()

1: $(pk, sk) \leftarrow$ FrodoPKE.KeyGen()
2: **return** $(pk, (pk, sk))$

Algorithm 5 FrodoKEM.Enc(pk)

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

Algorithm 6 FrodoKEM.Dec($(pk, sk), c$)

1: $\mu \leftarrow$ FrodoPKE.Dec(sk, c)
2: $r \leftarrow H(\mu)$
3: $c' \leftarrow$ FrodoKEM.Enc$(pk, \mu; r)$
4: **if** $c' = c$ **then**
5: **return** μ
6: **else**
7: Decapsulation fails.

2.4 The Rowhammer Bug

Rowhammer is a phenomenon wherein repeated accesses to a row in DRAM can induce bit flips in the neighboring rows [34, 43]. This is because activations of the row's wordline cause the capacitors storing bit values in neighboring rows to discharge slightly due to parasitic current. If this occurs a sufficient number of times to drop the voltage below the "charged" threshold before the DRAM refreshes, which typically occurs every 64ms, the logical value of the bit flips. The row that is repeatedly activated, or "hammered" is called the "aggressor row," while the rows containing the induced bit flips are "victim" rows.

Double-Sided Rowhammering. In order to use Rowhammer to intentionally flip inaccessible bits, it is more effective if the attacker repeatedly triggers accesses to memory values both *above and below* the victim rows within the same bank. If the victim row contains bits susceptible to Rowhammer, this memory access pattern is far more likely to induce bit flips than single sided hammering.

Repeatability of Flips. A crucial property of Rowhammer for building attacks is that Rowhammer-induced bit flips are repeatable. This means that a bit that flips at any given point in time is likely to flip again in the future when hammered, and similarly bits that do not flip after being hammered are unlikely to flip upon further hammering. While roughly half of flippable bits can flip in the 1-to-0 direction, and the other half in the opposite direction, any given bit can only possibly flip in one direction per boot. This means that an adversary can "profile" a given machine prior to an attack by hammering memory and building a map of where the flippable bits are and what directions they flip. This precision allows us to accurately poison the FrodoKEM key in a fairly deterministic manner.

Rowhammer History. A multitude of works from both academia and industry have helped Rowhammer evolve from its conception as a theoretical attack to a realistic attack vector with serious implications. After Kim et al. [34] first uncovered the Rowhammer phenomenon, researchers primarily focused on how to use Rowhammer for privilege escalation attacks and obtaining arbitrary read/writes [10, 22, 27, 37, 56, 58, 65, 66, 68]. Rowhammer attacks from the browser [18, 29], over the network [39, 63], and against ECC memory [14] soon followed, along with new hammering patterns[23, 33, 40] enabling Rowhammer against DDR4 memory.

Cryptographic Applications of Rowhammer. In addition to compromising standard isolation primitives, Rowhammer-induced bit flips were also used to break the security of cryptographic primitives. More specifically, Razavi et al. [54] demonstrate how Rowhammer can be used to break RSA signature validation, while Mus et al. [42] demonstrate an attack on the LUOV signature scheme. Finally, Kwong et al. [37] show how to use Rowhammer-induced bit flips to directly read bits from memory, allowing for the recovery of RSA keys directly from the target's address space.

3 DECRYPTION FAILURE ATTACK ON ROWHAMMER-POISONED FRODOKEM

In this section, we assume the reader has familiarity with decryption failure attacks (see Section 1.2) and Rowhammer attacks (see Section 2.4). It will also be helpful for the reader to reference Section 1.2 for an overview of our attack and our high-level strategy of combining a decryption failure and Rowhammer attack.

Given the above background, we begin by highlighting an important difference between our attack and a traditional decryption failure attack [7, 8, 16, 17, 19–21, 51]: In a traditional decryption failure attack, the failing ciphertexts contain a significant amount of information on the secret key, so that only a few thousand failures are required for a full key recovery. Because our effective failure threshold is so much lower, the failing ciphertexts generated using our Rowhammer-altered public key contain less information. The failure event occurs orders of magnitude more often for randomly generated ciphertexts, and in turn, the fraction of secret keys that are consistent with a single failure event is far higher. This induces a trade-off in which we require significantly more failing ciphertexts to gain enough information to recover the key, but require less computation *overall*, since it is much easier to find failing ciphertexts. It is important to quantify this trade-off, as it informs which specific bits of the public key to target with Rowhammer. First, we will analyze the effects of Rowhammer on the decryption failure rate, which gives us an estimate for the total amount of work required to generate a *single* failing ciphertext.

3.1 Decryption Failure Rate Analysis

The decoding error in Frodo640 is an 8×8 matrix

$$\mathbf{E}''' = \mathbf{S}'\mathbf{E} - \mathbf{E}'\mathbf{S} + \mathbf{E}''. \tag{1}$$

The decoded message, which is also an 8×8 matrix, is correct in each of its entries if the corresponding entry in $\mathbf{E}'''$ mod q is in the interval $[-4096, 4096)$. Otherwise, the decoded entry is incorrect.

Consider the entry $e'''_{i,j}$ at position (i, j) of $\mathbf{E}'''$. Let $\mathbf{s}'_i$ and $\mathbf{e}'_i$ be the i-rows of $\mathbf{S}'$ and $\mathbf{E}'$ respectively, and $\mathbf{e}^j$ and $\mathbf{s}^j$ be the j-columns of $\mathbf{E}$ and $\mathbf{S}$ respectively. Then,

$$e'''_{i,j} = \mathbf{s}'_i \cdot \mathbf{e}^j - \mathbf{e}'_i \cdot \mathbf{s}^j + e''_{i,j}.$$

Let χ be the error distribution in Frodo. If key generation is executed correctly, $\mathbf{e}^j$ is sampled from χ^n. However, due to row-hammering, the true distribution of $\mathbf{e}^j$ is $\chi' \times \chi^{n-1}$, where χ' is defined as follows. Let $p_\chi : \mathbb{Z} \to \mathbb{R}$ be the density function of χ. Then, $p_{\chi'}$ is defined by

$$p_{\chi'}(x) = \begin{cases} p_\chi(x) & \text{if } -12 \le x < 0 \\ p_\chi(x-256) & \text{if } 256 \le x < 269 \\ 0 & \text{otherwise} \end{cases}.$$

For Frodo640, the support of p_χ is $\{-12, \dots 12\}$ whereas the support of $p_{\chi'}$ is $\{-12, -11, \dots, -1, 256, 257, \dots, 268\}$. The reference implementation of FrodoKEM uses 16-bit integers. Here, we are targeting an attack that only requires that one bit flip per column of $\mathbf{E}$, since requiring more bit flips can reduce the success of the Rowhammer attack. By forcing the eighth-order bit of an entry to 1, the Rowhammer attack effectively adds 256 to any positive value and leaves negative values unchanged. Thus explaining the distribution of χ' shown above. Indeed, this analysis can be repeated for other orders of bits. We also consider the failure probabilities for a sixth-order (64) bit Rowhammer and a seventh-order (128) bit Rowhammer.

With $\mathbf{e}_j$ sampled from $\chi' \times \chi^{n-1}$, an honest encapsulation has decoding error distributed as

$$e'''_{i,j} \sim \chi \cdot \chi' + (\chi \cdot \chi)^{\circledast(2n-1)} + \chi, \tag{2}$$

where the notations are as follows. If χ_1 and χ_2 are two distributions, then $\chi_1 \cdot \chi_2$ is the distribution of the product of 2 independent random variables having distributions χ_1 and χ_2 respectively. Moreover, $\chi_1 + \chi_2$ is the convolution of χ_1 and χ_2, which is the distribution of the sum of 2 independent random variables having distributions χ_1 and χ_2 respectively. Finally, $\chi_1^{\circledast k} := \underbrace{\chi_1 + \chi_1 + \cdots + \chi_1}_{k \text{ times}}$. Concretely, by explicitly computing the distribution in equation (2), we have that such $e'''_{i,j}$ causes decryption failure with probability approximately $2^{-27.8}$ ($2^{-113.2}$ for a 64-bit Rowhammer, and 2^{-76} for a 128-bit Rowhammer).

For our adversarial attack, $\mathbf{s}'_i$ is not honestly sampled from χ^n. Instead, we filter the output of the random oracle and select only the values $\mathbf{s}'_i$ that have ± 12 in the same coordinate where χ' is in $\mathbf{e}^j$. Let τ be the distribution with probability $1/2$ at 12 and -12. Then, our adversarial $e'''_{i,j}$ has the distribution

$$e'''_{i,j} \sim \tau \cdot \chi' + (\chi \cdot \chi)^{\circledast(2n-1)} + \chi.$$

Concretely, such $e'''_{i,j}$ exceeds the decoding threshold of Frodo640 with probability approximately 2^{-13} ($2^{-98.7}$ for a 64-bit Rowhammer, and 2^{-61} for a 128-bit Rowhammer). To obtain one such $\mathbf{s}'_i$, the expected number of calls to the random oracle is 2^{15}. If there are c target columns of $\mathbf{E}$, since there are 8 rows of $\mathbf{S}'$, the expected number of calls per target column is $2^{15}/(8c)$. In our experiment, $c = 7$.

We note that there is a large variation in decryption failure rates conditioned on whether χ' is negative. The decryption failure rates conditioned on χ' being negative are negligible. Therefore, the decryption failure rates calculated above are negligibly different from decryption failure rates conditioned on χ' being positive. So we can mount our attack to recover $\mathbf{e}^j$ and $\mathbf{s}^j$ with probability $\Pr[\chi' > 0] = \Pr[\chi \ge 0]$.

Next we will describe our key recovery procedure, and analyze the number of failing ciphertexts necessary for recovering (each column of) the secret.

3.2 FrodoKEM Key recovery

Each failing entry of the error matrix in (1) gives rise to a linear inequality involving a column from each of $\mathbf{S}$, $\mathbf{E}$, a row from each of $\mathbf{S}'$, $\mathbf{E}'$, and the matching coordinate from $\mathbf{E}''$. Let $e'''_{i,j}$ be a failing entry of the error matrix. Assuming we exceed the failure threshold t in the positive direction, we obtain the following linear inequality

$$\begin{aligned} e'''_{i,j} = \mathbf{s}'_i \cdot \mathbf{e}^j + e''_{i,j} - \mathbf{e}'_i \cdot \mathbf{s}^j \ge t \\ (\mathbf{s}'_i \| - \mathbf{e}'_i) \cdot (\mathbf{e}^j \| \mathbf{s}^j) \ge t - e''_{i,j} \end{aligned}$$

Let us first consider a standard decryption failure attack with adjusted failure threshold $(t - e''_{i,j})$. In this attack, the adversary generates honestly distributed ciphertexts by running the encryption algorithm (this is enforced in practice by the Fujisaki-Okamoto transform) and queries them to the decryption oracle. The attacker then collects all the ciphertexts that led to decryption failure. We will first show how to perform key recovery in the above setting. We will then explain why our Rowhammer attack essentially reduces to a standard decryption failure attack with a further adjusted failure threshold $(t' - e''_{i,j})$, where $t' = t - 12 \cdot e_{i,j}$, and one fewer dimension.

In the above attack, $\mathbf{S}'$ and $\mathbf{E}'$ that produce failing ciphertexts are correlated with $\mathbf{E}$ and $\mathbf{S}$ respectively. Given a set of failing ciphertexts, there are various ways to use the correlation between the failing ciphertexts and the LWE secrets to learn information about the LWE secret. For example, D'Anvers et al. [16] use the correlation to calculate explicit posterior probabilities for the values of the secrets. These then allow them to calculate revised distributions of the secret with a smaller variance.

The posterior probabilities calculated as in D'Anvers et al. [16] are difficult to analyze. We therefore instead consider the distribution of failing ciphertexts as in [15]. This distribution is parametrized by the LWE secret itself, which is, of course, unknown. We then use our supply of failing ciphertexts–which constitute random samples from this distribution–to learn the hidden parameters. Specifically, as observed in [15], failing ciphertexts can be decomposed into a single component, a, distributed as a truncated, univariate Gaussian in the direction of the secret, and components distributed as independent Gaussians in the directions orthogonal to the secret.

Note that for FrodoKEM, the secret and error distributions are discrete approximations of spherical Gaussian distributions. However as we will elaborate on below, the distribution of the failing ciphertexts can be very well approximated as a (continuous) spherical multivariate Gaussian distribution with known variance and unknown mean (where the mean depends on the LWE secret). We can thus reframe the key recovery problem as the problem of estimating the mean of a multivariate Gaussian distribution with a known covariance matrix, given samples from that distribution.

Modeling Failing Ciphertexts More formally, let us assume that $\mathbf{s}, \mathbf{e}, \mathbf{s}', \mathbf{e}'$, and e'' are as above, are all drawn coordinate-wise from

a Gaussian distribution with zero mean and covariance σ_{se}^2. We omit the explicit row and column coordinates to reduce notational clutter, as the following holds regardless of the location of the failing coordinate in $\mathbf{E}'''$. Then, let us assume the norm of $(\mathbf{e}\|\mathbf{s})$ is exactly $\ell = \sqrt{n}\sigma_{se}$, where n is the dimension of $(\mathbf{e}\|\mathbf{s})$. This assumption is rather innocuous due to the high concentration of the norm of a Gaussian vector. As $(\mathbf{s}'\| - \mathbf{e}')$ is the part of the failing ciphertext that induces the failure condition, we can condition its distribution on the learned linear inequality

$$(\mathbf{s}'\| - \mathbf{e}') \sim \mathcal{N}(\mathbf{0}, \sigma_{se}^2 \mathbf{I}_n) \mid (\mathbf{e}\|\mathbf{s}) \cdot (\mathbf{s}'\| - \mathbf{e}') \geq t - e''$$

After conditioning, $(\mathbf{s}'\| - \mathbf{e}')$ decomposes into $(\mathbf{s}'\| - \mathbf{e}') = \frac{a}{\ell}(\mathbf{e}\|\mathbf{s}) + \mathcal{W}'$ where $\mathcal{W}'$ is a random vector distributed as a Gaussian of covariance $\sigma_{se}^2 \mathbf{\Pi}^{\perp}_{(\mathbf{e}\|\mathbf{s})}$ (i.e. independent noise in all directions orthogonal to the secret) and a is a random variable that is independent of $\mathcal{W}'$ and follows a distribution that we denote $\mathcal{N}^{\geq t/\ell}_{\sigma_{se}}$—the univariate Gaussian of variance σ_{se}^2 conditioned on $a \geq (t - e'')/\ell$.

In essence, each failing ciphertext gives a draw from the decomposed distribution. If we scale each sample by $\frac{\ell}{a}$, we can simulate draws from the distribution

$$\mathcal{D} := (\mathbf{e}\|\mathbf{s}) + \mathcal{W}''$$

where $\mathcal{W}'' = \frac{\ell}{a}\mathcal{W}'$. However, we do not know the value of a for a given failing ciphertext. Instead, we will instead make use of the following heuristic. We know that $a \geq (t - e'')/\ell$. Thus, we fix $\alpha = (t - e'')/\ell$ for a, which introduces additional noise into the distribution of $\mathcal{W}''$. If we substitute α for the true mean of a in the calculation of the variance, we obtain

$$\mathbb{E}_{\chi \leftarrow \mathcal{N}^{\geq t/\ell}_{\sigma_{se}}}[((t - e'')/\ell - \chi)^2]$$

This quantity can be shown to be $\leq \sigma_{se}^2$ for any $(t - e'')/\ell \geq 0$. As such, we assume the additional noise is also a Gaussian with mean $\mathbf{0}$ and variance σ_{se}^2.

Thus, we approximate the total noise $\mathcal{W}''$ as a zero-centered multivariate Gaussian with a coordinate-wise variance of at most

$$(\ell/a)^2 = (\ell^2/(t - e''))^2 \sigma_{se}^2 = n^2 \sigma_{se}^6/(t - e'')^2.$$

Recovering the secret can be performed by obtaining enough failing ciphertexts to estimate the mean of $\mathcal{D}$.

To estimate the mean, we can simply draw sufficient samples from $\mathcal{D}$, take their average, and round coordinate-wise to the nearest integer. After averaging m failing ciphertexts, we obtain a sample from the distribution

$$\mathcal{D}' := (\mathbf{e}\|\mathbf{s}) + \mathcal{W}''_{(m)},$$

where the error $\mathcal{W}''_{(m)}$ is again a Gaussian and the variance of each coordinate is at most $n^2\sigma_{se}^6/(t^2 m)$. Note here that after averaging, the effects of individual e'' on $\mathcal{D}'$ can be ignored as e'' is zero-centered. For the key recovery to succeed, we require the magnitude of each error coordinate to be less than 0.5. Thus, for a given value of m, the success probability of the attack is

$$\begin{aligned} P(\text{Success}) &= P(|w''_{(m)i}| < 0.5 \ \forall \ i) \\ &= P(|w''_{(m)i}| < 0.5)^n \\ &= (1 - 2 \cdot P(w''_{(m)i} < -0.5))^n \\ &= \left(-\mathtt{erf}\left(\frac{-0.5 \cdot t\sqrt{m}}{\sqrt{2} \cdot n\sigma_{se}^3}\right)\right)^n \end{aligned} \tag{3}$$

Given a target success probability, we then solve for m numerically. Note that it is certainly possible to compute a more accurate value for a numerically, and use it in the calculation of the mean of $\mathcal{D}'$. In a practical attack scenario, both $\alpha = (t - e'')/\ell$ and $\mathbb{E}[a]$ should be used to calculate the candidate secret, as the specific values of the true secret (e.g. if it has a higher than expected norm) can affect the number of failing ciphertexts required when using the heuristic value for a. Using the heuristic for estimation of success probability is preferred, as it is more conservative (i.e. a smaller a results in a larger variance for $\mathcal{W}''$).

Comparison with the 'Hint' framework of [15] We note that decryption failure attacks can also be handled by the DBDD and 'hint' framework of [15], where a single failing ciphertext is viewed as a full dimensional approximate hint on the LWE secret/error. In general, integrating full dimensional approximate hints in the framework of [15] requires explicit inversion of matrices defined over the rationals, which is computationally prohibitive. Due to this, Dachman-Soled et al. provided a lightweight version of their framework, which allows one to estimate the concrete hardness of performing a lattice reduction-based attack in the decryption failure setting. However, the lightweight version is not suitable for our purposes here since we are interested in a full key recovery, not just hardness estimation. We attempted to use the full hints framework of [15] since the matrices that needed to be inverted were all diagonal. Therefore, integrating a single full dimensional approximate hint could be performed reasonably quickly. However, as discussed previously, our attack requires significantly more failing ciphertexts than a traditional decryption failure attack (on the order of 2^{17} failing ciphertexts see Figure 1). This additional factor resulted in the computational cost of using the full framework being untenable. An improved implementation, using parallelism or accelerators would be required to be competitive with our simple average-then-round approach.

3.3 Attack Modifications for Rowhammer Assisted Failures

We next explain why our Rowhammer-assisted attack can be viewed as a standard decryption failure attack with a lower threshold and one fewer dimension. More precisely, the exact Rowhammer pattern alters the effective decryption failure threshold. In our attack, one coordinate—$e_{i,j}$—in each column of $\mathbf{E}$ is increased by 256. To increase the chances of failure, we also search for 12s in the corresponding coordinates of $\mathbf{S}'$. This lowers the failure threshold for the remaining coordinates by $12 \cdot (256 + \bar{e}_{i,j})$, where $\bar{e}_{i,j}$ is the original, unhammered value of $e_{i,j}$. We denote this new, smaller threshold as $t' = t - 12 \cdot e_{i,j} = t - 12 \cdot (256 + \bar{e}_{i,j})$.

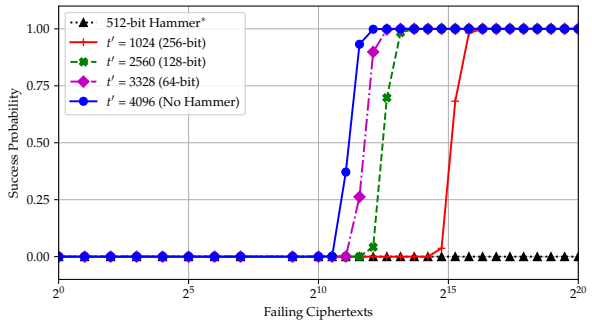

Figure 1: Probability of recovering a single column of the Frodo-640 secret (Equation 3), for given numbers of failing ciphertexts. Each line denotes a different (adjusted) threshold t' corresponding to rowhammer bit position.

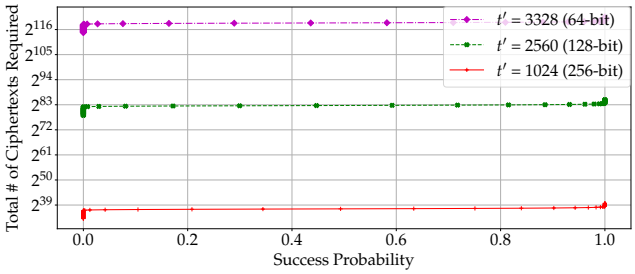

Figure 2: Total number of ciphertexts during failure-boosting (not decryption queries) required to recover a single column of the Frodo-640 secret, for (adjusted) thresholds t' corresponding to rowhammer bit position.

Note that since we fix one coordinate of $\mathbf{s}'$, the distribution of that coordinate does not depend on the LWE secret as described in the decomposition given in 3.2. As such, we simply ignore that coordinate when performing the averaging, thereby reducing the dimension by one. To calculate the proper value for $t' - e''_{i,j}$, we must guess this value in advance for each column. As the error distribution for Frodo-640 has 25 possible values, we simply try all values for $\bar{e}_{i,j}$, and check our recovered key using the LWE equations. The same failing ciphertexts can be used for each candidate value of t', so only the averaging step (which is computationally trivial) needs to be run 25 times. See Figure 1 for the relationship between the failure threshold $t' - e''_{i,j}$, and the number of failing ciphertexts m required to mount our attack with various success probabilities. Approximately $100k$ failing ciphertexts are required to succeed with probability close to 1 for a 256-bit rowhammered column. Note that flipping the 512-bit of an entry in $\mathbf{E}$ results in a completely unrecoverable secret, as decryption failure is virtually guaranteed assuming the same filtering behavior.

3.4 Total Attack Cost

We present the cost of the attack in the total number of ciphertexts (both succeeding and failing) needed to recover a single column of the secret key. We combine the prior analysis in 3.1 and 3.2 for 64, 128, and 256-bit rowhammers to produce the graph seen in Figure 2. From this, we can conclude that it is indeed best to target the 256-bit position with the rowhammer. The total number of ciphertexts we require to generate under failure-boosting to succeed with probability close to 1 is 2^{39} for the 256-bit rowhammer, compared to 2^{86} for the 128-bit rowhammer.

4 POISONING HOBBITS: ROWHAMMERING A FRODOKEM PUBLIC KEY

As outlined above, the main idea of our attack is to "poison" FrodoKEM's public key generation using Rowhammer. More specifically, we aim to flip specific bits inside the error term $\mathbf{E}$ computed during FrodoKEM's public key generation, obtaining a public key with a higher error. For FrodoKEM-640, the error term is an array of 640 x 8 values where each value is 2 bytes. Each of the values in the error matrix $\mathbf{E}$ has about an equal likely chance to be either positive or negative. By adding values to known locations in each of the matrix's columns, an attacker can increase the key's decryption failure rate. Finally, using the knowledge of flipped bit positions, the attacker can feasibly find a large number of failing ciphertexts, thereby enabling cryptanalytic attacks on FrodoKEM and ultimately recovering the secret key.

A Balancing Act. While the above description is conceptually simple, we note that to extract the secret key in practice, a delicate balance of Rowhammer-induced bit flips is required to execute our attack. More specifically, too many high-order bit flips in the error term boosts the decryption failure rate to an excessive rate, causing failures to be prevalent for both the attacker and honest users. This will alert the target to the possibility that they are under attack, causing them to retire the poisoned key. On the other hand, too few bit flips will cause the decryption failure to be too low, giving the attacker only a slight advantage over honest decryption failures. This will result in an infeasible amount of computation to determine the key exchanged by the KEM.

4.1 Experimental Setup

Unless noted otherwise, we performed all of our experiments using a desktop containing a 3.4 GHz i7-4770 CPU and two Samsung DDR3 4 GiB 1333 MHz non-ECC DIMMs (part number M378B5273CH0-CH9). Out machine was running a fully updated Ubuntu 20.04. To simplify the attack, we disabled the machine's address-space layout randomization (ASLR). We note that ASLR was breached on numerous prior occasions [13, 26, 28, 62] making this assumption not overly-restrictive.

The victim process is the FrodoKEM reference code, compiled using the reference optimization level and Shake-128 flags. Additional code was added to FrodoKEM to output files for the public key, secret key, and error matrix. These files were used for confirming the results of the attack. We took care to add these additional lines of code only after the attack was complete, to not artificially increase the window of opportunity for the Rowhammer attack.

4.2 Determining Useful Bit Flip Locations

Prior to mounting a Rowhammer attack, we must first determine which bits inside FrodoKEM's error matrix $\mathbf{E}$ are useful for the attacker to flip. To that aim, we used code provided in the FrodoKEM

submission package in order to examine how specific combinations of bit flips affect the decryption failure rate. As shown in Figure 1, we found that flipping bit 8, henceforth called a 256-bit flip because it adds $2^8 = 256$ to the value, in one value per column increased the decryption failure rate to about 2^{-27}, compared to a failure rate of $2^{-138.7}$ mentioned in Frodo-640's specification [3]. This achieves a nice balance of minimizing the number of required bit flips, while still raising the decryption failure rate enough to ensure the feasibility of our attack.

Given that there are 8 columns, the attack then requires a total of 8 256-bit flips in order to fully recover the secret key. Without all 8 flips, parts of the secret key must be recovered through a form of brute force search.

4.3 Profiling Memory

Prior to starting the Rowhammer attack itself, we must profile the machine's memory in order to locate the physical locations of pages containing bits susceptible to Rowhammer. We accomplished this using the memory massaging techniques described by Kwong et al. [37]. More specifically, the method of [37] involves exploiting Linux's buddy allocator to obtain 2 MB of contiguous memory regions required for precise double-sided Rowhammering. Unfortunately, following the disclosure of [37], modern Linux versions restrict access to the `/proc/pagetypeinfo` file used by [37] to root users. While this does required us to use elevated privileges to run our attack, a concurrent work by Tobah et al. [65] demonstrated how to perform the same attack on the allocator by using the world readable `/proc/buddyinfo` file, avoiding the use of root privileges.

After attacking the allocator to obtain tuples of memory locations for double-sided rowhammering, we iteratively hammer each pair of aggressor rows and then check the victim row for bit flips. Since bit flips are repeatable, we store the map of which bit flips were within each page so that we can select which pages are best to use for the attack.

Filtering Candidate Pages. For FrodoKEM-640, the error matrix has dimensions 640 x 8 of 2 bytes values, resulting in the matrix spanning 2.5 pages of size 4 KiB. Therefore, the error matrix can either be spread across 3 or 4 pages, depending on whether the array's page offset is past 0x800. Because of this, there must be multiple columns containing a 256-bit flip per page to satisfy all 8 required flips. Additionally, the first and last page storing the error matrix also store other nearby variables in the program; on our victim, this ended up being the S matrix and the array for storing randomness, respectively. While it may be possible to tolerate bit flips in S, we only considered pages containing no flips in these locations.

When choosing which 3 pages to use for our attack, we also must filter out both pages that have insufficient bit flips in the desired locations, and pages that have too many bit flips in undesired locations. Firstly, we filter out all sets of pages that don't provide exactly one 256-bit flip in at least 7 of the 8 columns. Otherwise, the Rowhammer attack will not increase the decryption failure rate enough for our attack to succeed. Equally important is that the pages do not exhibit bit flips during the attack in locations that will increase the decryption failure rate to be *too high*. For our attack, this means that any bit flips in bit positions 9 through 15, henceforth referred to as *high-order* bit flips, are not allowed. This is because increase in the error term will be very large in magnitude, and have an overly strong affect on the decryption failure rate. Additionally, only a small number of bits in positions 5 through 7 can be tolerated, since they will also have a substantial effect on the decryption failure rate, albeit less than the higher order bit flips. All other lower bits have an insignificant effect on the decryption failure rate.

Bit Suppression. With regards to requiring a bit to not flip, this can be accomplished by either choosing a page that doesn't contain a bit flip in that location, or by *suppressing* the undesired bit flip. As noted by Cojocar et al. [14], *stripe patterns*, where the bits above and below a given bit are the opposite value (i.e. 0-1-0 and 1-0-1 configurations), are the most likely to yield bit flips, while *uniform patterns* (i.e. 0-0-0 or 1-1-1) are unlikely to result in bit flips. Since bits can only ever flip in one direction, this means that we can use 1-1-1 patterns to suppress any 1-to-0 bit flip, and 0-0-0 patterns to suppress 0-to-1 bit flips.

By using the appropriate uniform patterns in the positions where the undesired bit flips are located, we significantly reduce the chance that those bit flips occur during the Rowhammer attack. We used this technique to suppress 117 bit flips in the chosen pages, leaving only 4 bits that could not be suppressed.

We note that not all undesired bit flips need to be suppressed, as there is a 50% chance that the randomly chosen value in **E** of the bit will already be the value that the bit can flip to. This means that for each unsuppressed bit flip, the probability of the attack succeeding decreases by a factor of 2. For example, if 3 bits cannot be suppressed, the attack succeeds 1 out of every 8 attempts.

4.4 Allocating Pages to Victim Process

After profiling the memory, the attacker must force the FrodoKEM victim to allocate its memory in a way that stores the secret term E in the attacker's chosen pages. This is accomplished through exploiting the Linux page frame cache in a technique termed "Frame Feng Shui" [37]. The Linux page frame cache stores frames in a first-in-last-out data structure, and returns the most recently deallocated page when receiving a request for a page frame. Therefore, if FrodoKEM allocates a predictable number of frames before the target **E**, the target can be forced into a page of our choice.

First, multiple 4 KiB junk pages are allocated using mmap and the MAP-POPULATE flag to fill the bottom of the page frame cache. The number of junk pages is determined by how many pages the victim process allocates before allocating the target value. Next, the attacker chooses pages from the profiling phase that contain vulnerable bits in the correct positions. The attacker then deallocates the selected page frames using `munmap`, putting them at the top of the page frame stack. Immediately after this, the attacker unmaps all of the previously mapped junk pages, putting these pages on top of the selected pages in the stack. Finally, the the attacker induces the victim process to run its key generation process, during which the FrodoKEM process requests memory to store E, and the buddy allocator returns the pages chosen by the attacker. We used Frame Feng Shui against the FrodoKEM-640 scheme with a total of 297 junk pages to land the E error matrix into the selected pages.

4.5 Performance Degradation

Now that the attacker has forced the victim FrodoKEM to allocate the selected pages for the error matrix, and the spatial precision of the bits is sufficient, we must now ensure that the temporal precision of the bit flips is sufficient. That is, the attacker must flip the bits within a precise window during the execution of the FrodoKEM key generation. By examining the source of the reference code, we identified that the bit flips must occur after the generation of E by sampling the Gaussian in the `crypto_kem_keypair` function, and before E is added to AS in the `frodo_mul_add_as_plus_e` function. In our setup, this window takes roughly 8ms, whereas we empirically found that we required 1300 ms to reliably succeed with the Rowhammer attack. To bridge this gap, we conducted a *performance degradation* attack against FrodoKEM.

A performance degradation attack [4] functions by rapidly forcing the most frequently executed FrodoKEM instructions to be evicted from the cache, which causes the CPU's pipeline to be flushed in a machine clear upon detecting the invalid tag in the L1i cache. This can result in dramatic decreases in performance if it occurs frequently enough. To evict the cacheline, we used the `CLFLUSH` instruction, available on x86 machines to all unprivileged users, to completely evict cachelines from the cache hierarchy.

Choosing a Cacheline. We statically analyzed the FrodoKEM victim's binary to find the most frequently executed cache line of code, and found that the usage of Shake-128 involves a tight loop that executes the `store64` instruction, resulting in 880992 calls to `store64` within the execution window.

This makes `store64` the optimal choice for performance degradation, so we implemented our performance degradation attack by running FrodoKEM-640 on a single core, with performance degraders running on both the sibling core and 2 other virtual cores, including the core running FrodoKEM-640. Without any performance degradation, the time hammering window is 8.2 ms. With performance degradation on only the sibling core, this can be increased to around 663 ms. With additional cores running the degrade code, this reached about 1300 ms. Using the performance degradation code with multiple cores, this allowed for sufficient time to complete the Rowhammer attack within the alotted window.

4.6 Results

For the attack against FrodoKEM-640, the page offset of the error matrix $\mathbf{E}$ was 0x9b0, meaning that it spanned a total of 4 pages. To prevent hammering any of the shared values on the first page, only the last 3 pages were targeted for the rowhammer attack. Using the criteria listed above, we found 3 pages that satisfy 7 of the 8 256-bit flips. Therefore, additional steps are required at the end for full key recovery. Also, bits in 4 matrix locations were found that could not be suppressed. Thus, accounting for the 7 256-bit locations that must be positive for the flip and the 4 locations that could not be suppressed, the attack succeeded 1 out of every 2^{11} attempts.

5 SEARCHING FOR FAILING CIPHERTEXTS WITH A SUPERCOMPUTER

In the construction of the poisoned public key, we intentionally keep the failure rate for honestly generated ciphertexts low so that the attack may persist for the duration of a generated key without the victim identifying the issue. This has the effect that finding a failing honestly generated ciphertext is still exceedingly rare. In the key established in Section 4, the failure rate of honestly generated ciphertexts is approximately 1:2,700,000. In order to generate approximately 100k failing ciphertexts per column, that would necessitate the creation of over 1 trillion connection requests. Alternatively, the adversary could generate ciphertexts that are likely to fail, but that would fail the Fujisaki-Okamoto transform.

To meet these constraints, our attack is predicated on a setup phase to generate honest ciphertexts that are *likely to fail* based on the profiled machine and the targeted columns. We use the $\mathbf{A}$ matrix from the public key, and then iterate on the random seed that initializes μ as input to the Frodo.Encode function. Ordinarily, μ is initialized by a randomly sampled 16-byte seed, and thus there are 2^{128} potential seeds. The encode function continues as specified until the $\mathbf{S}'$ matrix is generated. The superset of the rows and columns targeted by Rowhammer are checked to determine if they have high values (i.e. -12 or +12) that would make them candidates to potentially fail to decode. These candidate ciphertexts are saved to attempt decoding. For the poisoned key established in section 4, otherwise honestly generated ciphertexts (i.e. those that pass the FO transform) pass this filter at a ratio of approximately 1:1175.

For our proof-of-concept attack, all candidate ciphertexts are passed as input to the Frodo.Decode function. Ciphertexts that fail are saved to a file including the μ seed to generate the ciphertext, the $\mathbf{S}'$, $\mathbf{E}'$ and $\mathbf{E}''$ matrices, and the ciphertext itself. The attacker would have access to all of these aspects of the ciphertext in a distributed attack. The components are concatenated to files named for the row/column pair where the high value in the $\mathbf{S}'$ matrix is located (e.g. 'mu-4-8.csv'). This helps to identify the likely column that created the failing condition, which aids the key-recovery process. So, we only considered matrices that had a single ±12 in the checked locations.

To expedite the profiling stage we leveraged resources from two supercomputer clusters — namely Pittsburgh Supercomputing Center (PSC) [12] and Open Science Grid (OSG) [49, 60]. Total ciphertext generation took 237,806 hours, distributed as follows: 104,856 core-hours on OSG through the Open Science Pool, and 132,950 core-hours on the Bridges-2 Regular Memory nodes from PSC. For PSC, the modified Frodo-KEM encode and decode functions were compiled directly on each compute node. For OSG, the modified code was compiled in a container for each compute node. Jobs were launched with an incrementing seed to generate 150 million ciphertexts per seed. The start seed was shifted 28 bits to the left and stored in μ as the start of the search. Each job produced approximately 128 thousand ciphertexts that passed the filter, from which approximately 20 ciphertexts failed to decrypt. In total 665,343 failing ciphertexts were used to recover 7 of the 8 columns, necessitating 1.53 billion decryption requests—less than 10% of the absolute maximum 2^{64} decryption queries identified by NIST to prevent failure attacks.

6 COMPLETING THE ATTACK: SESSION-KEY RECOVERY

If the full master secret key is unable to be recovered, it is still possible to recover encapsulated session keys from honestly generated

Algorithm 7 SessionKeyBF(**SExp**, $pk, col, ct = (ct_1, ct_2)$)

$\mathbf{C_1}$:= FrodoKEM.unpack(ct_1)
$\mathbf{C_2}$:= FrodoKEM.unpack(ct_2)
Compute $\mathbf{M'} = \mathbf{C_2} - \mathbf{C_1} \cdot \mathbf{SExp}$
μ' := FrodoKEM.Decode($\mathbf{M'}$)
for $i := 0 \rightarrow 2^{16}$ **do**
    Insert(μ', col, i) ▷ Insert i into col's bit positions in μ'.
    ct', ss := FrodoKEM.Encaps(pk, μ')
    if $ct' = ct$ **then return** ss
return FAIL

Figure 3: Session key brute-force algorithm given the experimentally derived secret and missing column index.

ciphertexts. The recovery procedure is enabled by the limited number of bits present in the message μ. Due to the Fujisaki-Okamoto transform, encapsulation is deterministic given the value of μ. If we can determine the value of μ, we can easily recover the session key.

In FrodoKEM.Decaps [3], μ is decoded from an 8x8 message matrix $\mathbf{M}$, which is computed from an honestly generated ciphertext and the secret key. As we are unable to compute $\mathbf{M}$ due to our lack of the full secret key, instead we compute $\mathbf{M'}$ by filling in the missing columns of the secret key with 0's in the computation of $\mathbf{M}$. Note that $\mathbf{M'} = \mathbf{M}$ except in the missing columns. For Frodo-640, the most significant 2 bits of each entry of $\mathbf{M}$ are extracted to produce μ in FrodoKEM.Decode (resulting in a message size of 128 bits). Thus, for every missing column of the secret key, we must brute-force 16 bits of μ. To check correctness, we simply pass each μ' to a modified version of FrodoKEM.Encaps that accepts μ as a parameter rather than sampling it at random. The deterministic nature of the encryption ensures that the output ciphertext, will be equal to the intercepted ciphertext from the victim when we pass in the correct value for μ.

The total cost of the session key recovery is $2^{16 \cdot \mathrm{missingcols}}$ times the cost of FrodoKEM.Encaps. In our attack, we managed to recover 7/8 of the columns of the secret key, and only needed to brute-force 16 bits. This procedure takes at most a couple of minutes on a commodity laptop. In a scenario where super computers are enlisted for this brute-force search in the online phase of the attack, more missing columns could be tolerated at the discretion of the attacker. Sessions could be captured in the meantime for later decryption once the brute-force completes.

7 ATTACKING OTHER NIST LATTICE KEMS

While we only experimentally demonstrated an attack against FrodoKEM, we believe that similar attacks are theoretically possible on other latice-based KEMs; however, there are various reasons these attacks may be more difficult in practice.

In this section we will look at the other lattice KEMs in the NIST PQC Round 3 candidate pool: Kyber, Saber, NTRU, sNTRUprime, and NTRU-LPRime. In all these cases, the strategy is similar: We note that decapsulation failures occur with high probability when the product between a noise vector chosen during key generation and a second noise vector chosen during encapsulation is large. The strategy is then to:

(1) Introduce known additional noise during key generation using Rowhammer techniques, producing a "poisoned key"
(2) Use knowledge of the additional noise to select valid ciphertexts that are likely (but not too likely) to produce a decapsulation failure when the honest party attempts to decrypt them using the "poisoned" key.
(3) Accumulate information about the unknown parts of the private key by observing which of the ciphertexts are and are not successfully decapsulated.

In order to accomplish step 1, it is likely that one would need to do performance degradation similar to what was done to FrodoKEM in our experiments. For Kyber and Saber, we identify places in the code where the SHAKE function is used between the time when noise is sampled and when it is used. Cryptographic random number generation is also done in sNTRUprime and NTRU-LPRime in a location that appears similarly useful. However, the reference implementation of each uses AES instead of SHAKE as the core primitive. Since AES has native hardware support, it is likely harder to do performance degradation against the reference implementation of NTRUprime. If the cryptographic random number generation could be performance degraded, sNTRUprime seems like a better candidate for attack than NTRU-LPRime, because in the latter case, the random number generator is called only once, while in the former case it is called repeatedly.

An additional difficulty arises when trying to adapt the attack to NTRU. In that case, the fact that algebraic operations switch between the rings $\mathbb{Z}_q[x]/(x^n - 1)$ and $\mathbb{Z}_q[x]/((x^n - 1)/(x - 1))$ means that any rowhammering which fails to preserve the sum mod q of the coefficients of a polynomial will cause decryption to fail with overwhelming probability for all ciphertexts. This makes it very likely that any attack will be detected, and means that decryption failures will give no information about the private key. It is unclear how to design an attack which avoids this issue.

With regard to step 2, the filtering step, there is an additional quantitatve challenge compared to our attack on FrodoKEM. In the case of FrodoKEM, we were able to Rowhammer a single bit per column of $\mathbf{E}$. This was possible because the parameters of the attack could be set so that a decryption failure only occurred with significant probability when one of 56 locations in the ciphertext noise attained a maximal value. Since this only happened with low probability for an honestly generated ciphertext, filtering was possible. In all the other KEMs, each position in the key generation noise multiplies more positions in the ciphertext noise, and the ciphertext noise distribution has the maximum value occur much more frequently. As a result we expect that attacking any of the other schemes will require a somewhat higher density of rowhammerable bits. The filtering criterion in this case would require the products of several coefficients of the ciphertext noise with the rowhammered coefficients to be large and to have the same sign. We give more details on the methods that would need to be employed to attack the other schemes in Appendix A.

8 POSSIBLE COUNTERMEASURES

As described, our attack shows that Frodo is vulnerable against a real-world Rowhammer attack. This justifies further study of the Rowhammer security of any post-quantum protocols that will come out of the NIST standardization process. As such countermeasures are critical, we wish to highlight a few possible ones up front.

8.1 Rowhammer Defenses

There is a rich literature surrounding Rowhammer defenses, both proposed and deployed, that attempt to mitigate the Rowhammer bug. Despite these efforts, researchers have managed to circumvent all practically deployed mitigations and empirically demonstrate bit flips against them. Even so, a combination of such countermeasures to provide "defense in depth" may dramatically increase the effort required to mount a Rowhammer attack.

Hardware Defenses. The two hardware defenses deployed in practice are Error Correcting Code (ECC) memory and the Targeted Row Refresh (TRR) mitigation. ECC memory, most commonly found on server machines, aims to correct bit flips when they are detected upon reading from memory. The memory controller stores additional "control bits" that act as a checksum, enabling bit correction and detection up to a certain threshold of errors. While the ECC mechanism was designed to mitigate errors due to cosmic rays, it also inadvertently assists in mitigating Rowhammer bit flips. [14] defeated ECC with Rowhammer, however, by causing a sufficient number of bit flips within an ECC word such that the mechanism can no longer even detect that a bit flip has occurred.

TRR is a hardware defense implemented in DDR4 that was long touted to be a panacea for Rowhammer. It uses counters to keep track of how many times each row in memory is accessed, and once the counter reaches a specified threshold, TRR automatically refreshes the nearby rows. If this threshold is below the minimum number of hammering attempts required for a Rowhammer attack, then this mitigation refreshes victim bits before they are hammered enough to flip. While this seems to be a perfect defense in theory, [23] were able to bypass TRR and flip bits on DDR4 due to limitations imposed by constraints in the hardware. In particular, TRR can only track accesses to a small number of rows at a time, and attackers can bypass it by hammering many rows at once.

Most recently, [40] addressed the shortcomings of TRR by proposing a scheme that tracks rows optimally, given a limited number of counters and additional refresh commands. While they demonstrate that their algorithm achieves the best trade-off between these parameters, it still suffers from the inherent limit to how many row counters can be supported by the hardware.

Software Defenses. While no software level defenses against Rowhammer have seen widespread adoption, researchers have proposed mitigations with claims of low overhead and complete Rowhammer protection. [6] aim to backport the same principles behind TRR to DDR3. They use performance counters to determine which rows are accessed most frequently, and then refresh those rows before a bit flip occurs.

Brasser et al. [11] aim to protect the kernel from user level Rowhammer attacks by physically separating the kernel memory on the DIMM from user memory. [36] go a step further and physically isolate all data rows in memory from each other.

8.2 Defenses Specific to our Attack

We also present practical countermeasures derived from unique insights gleaned from our end-to-end attack against FrodoKEM.

Hardware Accelerated Cryptography. As demonstrated in Section 4, having a sufficiently long time window to rowhammer is critical to the success of the attack. To extend the length of this window all the way from 8ms to 1300ms, we relied on a performance degradation attack against the SHAKE hash function, wherein we rapidly flushed a cacheline containing frequently executed FrodoKEM code. The effectiveness of the performance degradation is highly dependent upon how many calls FrodoKEM makes to the flushed cacheline – the one containing `store64` in this case.

If, however, FrodoKEM were to use hardware accelerated cryptographic instructions in place of its in-software hash function implementation, the opportunity for the attacker to conduct a performance degradation attack would be greatly diminished. This is because the tight inner loop that calls `store64` within Shake would instead be replaced, for example, by just a few instructions from Intel's AES-New Instructions (AES-NI) instruction set. We thus observe that hardware accelerated cryptographic functions seem to be more resilient against performance degradation attacks.

Algorithmic Defenses. In addition, there are two potential algorithmic defenses to our attack. The first is to limit the surface of the performance degradation through proper sequencing of operations, even when not using AES-NI. The main idea is to not perform any expensive operations between the time when **S** and **E** are sampled and when **B** is generated. For example, one could use SHAKE to expand out **A** before **S** and **E** are generated (like in Kyber and Saber) and move the SHAKE calls used to generate the randomness for **S** and **E** before the actual sampling (like in FrodoKem and Saber). Ordering the operations in this manner significantly reduces the window for performance degradation, and therefore our attack.

The second defense is to guard the key generation process. One way to do this is to regenerate **A**, **S**, and **E** from randomness and compute **B** again. If the two **B** are not equivalent, then abort key generation. For the Rowhammer attack to pass this check, the entire attack must succeed twice in a row, for potentially different pages of memory. In addition, to ensure the equality of the **B** matrices, the Rowhammer attack must flip *exactly* the same bits each time. It is highly unlikely, however, that the attacker would be able to find another set of pages that also exhibits the exact same bit flips under hammering as the original set.

Storing the error matrix E (Frodo overwrites E with B for efficiency) for the duration of the computation of the public key is another possible way to enforce the integrity of key generation. Then, if any value in E (or perhaps its distribution) is abnormal, recompute the public key from scratch. This has the downside of potentially alerting the attacker to the re-keying process, and enabling them to try to rowhammer the new key.

Active Defenses. Another line of defense would be to maintain an active state during the online phase to detect an attack in progress. An adversary needs to send a vast number of filtered ciphertexts to mount a successful key recovery attempt. A potential victim could check the distribution of received ciphertexts during the decapsulation process. Indeed, during the re-encryption check, the victim would be able to see the values of the randomness used during encapsulation (at least where decapsulation would ordinarily

be successful). If the distribution of incoming ciphertexts is skewed towards large values (e.g. ±12), then it is reasonable to assume that the victim is under attack. One must be careful in how to respond to this attack, however. If the decision is to simply re-key when an attack is detected, this leads to an easy Denial of Service attack (an attacker can perform the same filtering procedure to intentionally trigger the detector). On the other hand, if the response is to discard the received ciphertexts that contain such large values then this can lead to discarding communication from honest users.

Also, while this countermeasure would make our attack harder, it would not make it very much harder – if the intensity of the rowhammering is modestly increased, the attacker can make the decryption failure on random ciphertexts sufficiently high as to not require ciphertexts to come from an unusual distribution. This would, however, result in a significantly higher decryption failure rate for honest parties, making the attack easier to detect.

ACKNOWLEDGMENTS

We thank David Andrews, Miaoqing Huang, and Sean Maurey for many useful discussions. This research was done using services provided by the OSG Consortium [50, 61], which is supported by the National Science Foundation awards #2030508 and #1836650. Michael Fahr Jr. and Alexander Nelson were funded by the U.S. Department of Commerce, National Institute for Standards and Technology under the cooperative agreement #60NANB20D016. Hunter Kippen was supported in part by the Clark Doctoral Fellowship from the Clark School of Engineering, University of Maryland, College Park. Hunter Kippen and Dana Dachman-Soled were supported in part by NSF grant #CNS-1453045 (CAREER), by financial assistance awards #70NANB15H328 and #70NANB19H126 from the U.S. Department of Commerce, National Institute of Standards and Technology, and by Intel through the Intel Labs Crypto Frontiers Research Center. Andrew Kwong and Daniel Genkin were supported by the National Science Foundation under award number #CNS-1954712, the Air Force Office of Scientific Research (AFOSR) under award number FA9550-20-1-0425, and a gift from Qualcomm. Arkady Yerukhimovich was supported in part by NSF grant #CNS-1955620. Daniel Apon's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions, or viewpoints expressed by the author.

REFERENCES

[1] National Security Agency. 2021. Frequently Asked Questions: Quantum Computing and Post-Quantum Cryptography. https://media.defense.gov/2021/Aug/04/2002821837/-1/-1/1/Quantum_FAQs_20210804.PDF

[2] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, and Yi-Kai Liu. 2022. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process.* Technical Report : NIST Internal Report (NISTIR) 8413. U.S. Department of Commerce, Washington, D.C. https://doi.org/10.6028/NIST.IR.8413

[3] Erdem Alkim, Joppe W. Bos, Léo Ducas, Patrick Longa, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Chris Peikert, Ananth Raghunathan, Douglas Stebila, Karen Easterbrook, and LaMacchia Brian. 2022. FrodoKEM: Practical quantum-secure key encapsulation from generic lattices. https://frodokem.org/

[4] Thomas Allan, Billy Bob Brumley, Katrina Falkner, Joop van de Pol, and Yuval Yarom. 2016. Amplifying Side Channels through Performance Degradation. In *Proceedings of the 32nd Annual Conference on Computer Security Applications* (Los Angeles, California, USA) *(ACSAC '16)*. Association for Computing Machinery, New York, NY, USA, 422–435. https://doi.org/10.1145/2991079.2991084

[5] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510. https://doi.org/10.1038/s41586-019-1666-5

[6] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. 2016. ANVIL: Software-based protection against next-generation rowhammer attacks. *ACM SIGPLAN Notices* 51, 4 (2016), 743–755.

[7] Aurélie Bauer, Henri Gilbert, Guénaël Renault, and Mélissa Rossi. 2019. Assessment of the Key-Reuse Resilience of NewHope. In *Topics in Cryptology – CT-RSA 2019 (Lecture Notes in Computer Science, Vol. 11405)*, Mitsuru Matsui (Ed.). Springer, Heidelberg, Germany, San Francisco, CA, USA, 272–292. https://doi.org/10.1007/978-3-030-12612-4_14

[8] Nina Bindel and John M. Schanck. 2020. Decryption Failure Is More Likely After Success. In *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, Jintai Ding and Jean-Pierre Tillich (Eds.). Springer, Heidelberg, Germany, Paris, France, 206–225. https://doi.org/10.1007/978-3-030-44223-1_12

[9] Jonathan Bootle, Claire Delaplace, Thomas Espitau, Pierre-Alain Fouque, and Mehdi Tibouchi. 2018. 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, 494–524.

[10] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2016. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. In *IEEE SP*.

[11] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. 2017. {CAn't} Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory. In *26th USENIX Security Symposium (USENIX Security 17)*. 117–130.

[12] Shawn T Brown, Paola Buitrago, Edward Hanna, Sergiu Sanielevici, Robin Scibek, and Nicholas A Nystrom. 2021. Bridges-2: A Platform for Rapidly-Evolving and Data Intensive Research. In *Practice and Experience in Advanced Research Computing*. 1–4.

[13] Claudio Canella, Michael Schwarz, Martin Haubenwallner, Martin Schwarzl, and Daniel Gruss. 2020. KASLR: Break it, fix it, repeat. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. 481–493.

[14] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. 2019. Exploiting correcting codes: On the effectiveness of ECC memory against Rowhammer attacks. In *IEEE SP*.

[15] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. 2020. LWE with Side Information: Attacks and Concrete Security Estimation. In *Advances in Cryptology – CRYPTO 2020, Part II (Lecture Notes in Computer Science, Vol. 12171)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 329–358. https://doi.org/10.1007/978-3-030-56880-1_12

[16] Jan-Pieter D'Anvers, Qian Guo, Thomas Johansson, Alexander Nilsson, Frederik Vercauteren, and Ingrid Verbauwhede. 2019. Decryption Failure Attacks on IND-CCA Secure Lattice-Based Schemes. In *PKC 2019: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part II (Lecture Notes in Computer Science, Vol. 11443)*, Dongdai Lin and Kazue Sako (Eds.). Springer, Heidelberg, Germany, Beijing, China, 565–598. https://doi.org/10.1007/978-3-030-17259-6_19

[17] Jan-Pieter D'Anvers, Mélissa Rossi, and Fernando Virdia. 2020. (One) Failure Is Not an Option: Bootstrapping the Search for Failures in Lattice-Based Encryption Schemes. In *Advances in Cryptology – EUROCRYPT 2020, Part III (Lecture Notes in Computer Science, Vol. 12107)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, Heidelberg, Germany, Zagreb, Croatia, 3–33. https://doi.org/10.1007/978-3-030-45727-3_1

[18] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. 2021. SMASH: Synchronized Many-sided Rowhammer Attacks from JavaScript. In *USENIX Security*. Paper=https://comsec.ethz.ch/wp-content/files/smash_sec21.pdfURL=https://comsec.ethz.ch/research/dram/smash Pwnie Nomination for the Most Underhyped Research.

[19] Jintai Ding, Saed Alsayigh, R V Saraswathy, Scott Fluhrer, and Xiaodong Lin. 2017. Leakage of signal function with reused keys in RLWE key exchange. In *2017 IEEE International Conference on Communications (ICC)*. 1–6. https:

//doi.org/10.1109/ICC.2017.7996806

[20] Jintai Ding, Scott R. Fluhrer, and Saraswathy RV. 2018. Complete Attack on RLWE Key Exchange with Reused Keys, Without Signal Leakage. In *ACISP 18: 23rd Australasian Conference on Information Security and Privacy (Lecture Notes in Computer Science, Vol. 10946)*, Willy Susilo and Guomin Yang (Eds.). Springer, Heidelberg, Germany, Wollongong, NSW, Australia, 467–486. https://doi.org/10.1007/978-3-319-93638-3_27

[21] Scott Fluhrer. 2016. Cryptanalysis of ring-LWE based key exchange with key share reuse. Cryptology ePrint Archive, Report 2016/085. https://eprint.iacr.org/2016/085.

[22] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Grand pwning unit: Accelerating microarchitectural attacks with the GPU. In *IEEE SP*. 195–210.

[23] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2020. TRRespass: Exploiting the Many Sides of Target Row Refresh. In *S&P*. https://comsec.ethz.ch/wp-content/files/trrespass_sp20.pdf Best Paper Award, Pwnie Award for the Most Innovative Research, Honorable Mention in IEEE MICRO Top Picks.

[24] Eiichiro Fujisaki and Tatsuaki Okamoto. 1999. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *Advances in Cryptology – CRYPTO'99 (Lecture Notes in Computer Science, Vol. 1666)*, Michael J. Wiener (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 537–554. https://doi.org/10.1007/3-540-48405-1_34

[25] Bundesamt für Sicherheit in der Informationstechnik. 2022. BSI TR-02102-1: "Cryptographic Mechanisms: Recommendations and Key Lengths" Version: 2022-1. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf

[26] Ben Gras, Kaveh Razavi, Erik Bosman, Herbert Bos, and Cristiano Giuffrida. 2017. ASLR on the Line: Practical Cache Attacks on the MMU.. In *NDSS*, Vol. 17. 26.

[27] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. 2018. Another flip in the wall of Rowhammer defenses. In *IEEE SP*. 245–261.

[28] Daniel Gruss, Clémentine Maurice, Anders Fogh, Moritz Lipp, and Stefan Mangard. 2016. Prefetch side-channel attacks: Bypassing SMAP and kernel ASLR. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 368–379.

[29] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2016. Rowhammer.js: A remote software-induced fault attack in JavaScript. In *DIMVA*. 300–321.

[30] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. 2017. A Modular Analysis of the Fujisaki-Okamoto Transformation. In *TCC 2017: 15th Theory of Cryptography Conference, Part I (Lecture Notes in Computer Science, Vol. 10677)*, Yael Kalai and Leonid Reyzin (Eds.). Springer, Heidelberg, Germany, Baltimore, MD, USA, 341–371. https://doi.org/10.1007/978-3-319-70500-2_12

[31] James Howe, Ayesha Khalid, Marco Martinoli, Francesco Regazzoni, and Elisabeth Oswald. 2019. Fault attack countermeasures for error samplers in lattice-based cryptography. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[32] Saad Islam, Koksal Mus, Richa Singh, Patrick Schaumont, and Berk Sunar. 2022. Signature Correction Attack on Dilithium Signature Scheme. *CoRR* abs/2203.00637 (2022). https://doi.org/10.48550/arXiv.2203.00637 arXiv:2203.00637

[33] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. 2022. Blacksmith: Scalable Rowhammering in the Frequency Domain. In *S&P*. Paper=https://comsec.ethz.ch/wp-content/files/blacksmith_sp22.pdfURL=https://comsec.ethz.ch/research/dram/blacksmith

[34] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 361–372. https://doi.org/10.1109/ISCA.2014.6853210

[35] Daniel Kirkwood, Bradley C. Lackey, John McVey, Mark Motley, Jerome A. Solinas, and David Tuller. 2015. Failure is not an Option: Standardization Issues for Post-Quantum Key Agreement. https://csrc.nist.gov/csrc/media/events/workshop-on-cybersecurity-in-a-post-quantum-world/documents/presentations/session7-motley-mark.pdf.

[36] Radhesh Krishnan Konoth, Marco Oliverio, Andrei Tatar, Dennis Andriesse, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. 2018. {ZebRAM}: Comprehensive and Compatible Software Protection Against Rowhammer Attacks. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 697–710.

[37] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. 2020. RAMBleed: Reading Bits in Memory Without Accessing Them. In *41st IEEE Symposium on Security and Privacy (S&P)*.

[38] Richard Lindner and Chris Peikert. 2011. Better Key Sizes (and Attacks) for LWE-Based Encryption. In *Topics in Cryptology – CT-RSA 2011 (Lecture Notes in Computer Science, Vol. 6558)*, Aggelos Kiayias (Ed.). Springer, Heidelberg, Germany, San Francisco, CA, USA, 319–339. https://doi.org/10.1007/978-3-642-19074-2_21

[39] Moritz Lipp, Michael Schwarz, Lukas Raab, Lukas Lamster, Misiker Tadesse Aga, Clémentine Maurice, and Daniel Gruss. 2020. Nethammer: Inducing rowhammer faults through network requests. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 710–719.

[40] Michele Marazzi, Patrick Jattke, Flavien Solt, and Kaveh Razavi. 2022. ProTRR: Principled yet Optimal In-DRAM Target Row Refresh. In *S&P*. Paper=https://comsec.ethz.ch/wp-content/files/protrr_sp22.pdfURL=https://comsec.ethz.ch/research/dram/protrr Patent pending, ETH Spark Award Nomination.

[41] Daniele Micciancio and Oded Regev. 2009. Lattice-based cryptography. In *Post-quantum cryptography*. Springer, 147–191.

[42] Koksal Mus, Saad Islam, and Berk Sunar. 2020. QuantumHammer: A Practical Hybrid Attack on the LUOV Signature Scheme. In *ACM CCS 2020: 27th Conference on Computer and Communications Security*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, Virtual Event, USA, 1071–1084. https://doi.org/10.1145/3372297.3417272

[43] Onur Mutlu and Jeremie S. Kim. 2020. RowHammer: A Retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 8 (2020), 1555–1571. https://doi.org/10.1109/TCAD.2019.2915318

[44] NIST. 2016. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf

[45] National Institute of Standards and Technology (NIST). 2022. Post-quantum cryptography - Round 3 submissions. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions

[46] National Institute of Standards and Technology (NIST). 2022. Post-quantum cryptography standardization. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization

[47] Chris Peikert. 2016. A Decade of Lattice Cryptography. *Foundations and Trends in Theoretical Computer Science* 10, 4 (2016), 283–424. https://doi.org/10.1561/0400000074

[48] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. 2017. 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*. 1843–1855.

[49] Ruth Pordes, Don Petravick, Bill Kramer, Doug Olson, Miron Livny, Alain Roy, Paul Avery, Kent Blackburn, Torre Wenaus, Frank Würthwein, et al. 2007. The open science grid. In *Journal of Physics: Conference Series*, Vol. 78. IOP Publishing, 012057.

[50] Ruth Pordes, Don Petravick, Bill Kramer, Doug Olson, Miron Livny, Alain Roy, Paul Avery, Kent Blackburn, Torre Wenaus, Frank Würthwein, Ian Foster, Rob Gardner, Mike Wilde, Alan Blatecky, John McGee, and Rob Quick. 2007. The open science grid. In *J. Phys. Conf. Ser. (78, Vol. 78)*. 012057. https://doi.org/10.1088/1742-6596/78/1/012057

[51] Yue Qin, Chi Cheng, Xiaohan Zhang, Yanbin Pan, Lei Hu, and Jintai Ding. 2021. A Systematic Approach and Analysis of Key Mismatch Attacks on Lattice-Based NIST Candidate KEMs. In *ASIACRYPT 2021, Tibouchi and H. Wang (Eds.)*. 92–121. https://doi.org/10.1007/978-3-030-92068-5_4

[52] Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. 2018. Side-channel assisted existential forgery attack on Dilithium-a NIST PQC candidate. *Cryptology ePrint Archive* (2018).

[53] Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. 2019. 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*. 427–440.

[54] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. 2016. Flip Feng Shui: Hammering a Needle in the Software Stack. In *USENIX Security*. 1–18.

[55] Oded Regev. 2005. On lattices, learning with errors, random linear codes, and cryptography. In *37th Annual ACM Symposium on Theory of Computing*, Harold N. Gabow and Ronald Fagin (Eds.). ACM Press, Baltimore, MA, USA, 84–93. https://doi.org/10.1145/1060590.1060603

[56] Google Research. 2021. Half-Double: Next-Row-Over Assisted Rowhammer. https://github.com/google/hammer-kit/blob/main/20210525_half_double.pdf

[57] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. 2018. Tightly-Secure Key-Encapsulation Mechanism in the Quantum Random Oracle Model. In *Advances in Cryptology – EUROCRYPT 2018, Part III (Lecture Notes in Computer Science, Vol. 10822)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, Heidelberg, Germany, Tel Aviv, Israel, 520–551. https://doi.org/10.1007/978-3-319-78372-7_17

[58] Mark Seaborn and Thomas Dullien. 2015. Exploiting the DRAM Rowhammer bug to gain kernel privileges. https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html.

[59] Johanna Sepulveda, Andreas Zankl, and Oliver Mischke. 2017. Cache attacks and countermeasures for NTRUEncrypt on MPSoCs: post-quantum resistance for the IoT. In *2017 30th IEEE International System-on-Chip Conference (SOCC)*.

IEEE, 120–125.
[60] Igor Sfiligoi, Daniel C Bradley, Burt Holzman, Parag Mhashilkar, Sanjay Padhi, and Frank Wurthwein. 2009. The pilot way to grid resources using glideinWMS. In *2009 WRI World congress on computer science and information engineering*, Vol. 2. IEEE, 428–432.
[61] Igor Sfiligoi, Daniel C Bradley, Burt Holzman, Parag Mhashilkar, Sanjay Padhi, and Frank Wurthwein. 2009. The pilot way to grid resources using glideinWMS. In *2009 WRI World Congress on Computer Science and Information Engineering (2, Vol. 2)*. 428–432. https://doi.org/10.1109/CSIE.2009.950
[62] Kevin Z Snow, Fabian Monrose, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, and Ahmad-Reza Sadeghi. 2013. Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 574–588.
[63] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Throwhammer: Rowhammer Attacks over the Network and Defenses. In *USENIX ATC*. https://comsec.ethz.ch/wp-content/files/throwhammer_atc18.pdf Pwnie Award Nomination for the Most Innovative Research.
[64] Mehdi Tibouchi and Alexandre Wallet. 2021. One bit is all it takes: a devastating timing attack on BLISS's non-constant time sign flips. *Journal of Mathematical Cryptology* 15, 1 (2021), 131–142.
[65] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G Shin. 2022. SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks. In *43rd IEEE Symposium on Security and Privacy (S&P)*.
[66] Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. 2016. Drammer: Deterministic Rowhammer attacks on mobile platforms. In *CCS*. 1675–1689.
[67] Ricardo Villanueva-Polanco. 2019. Cold Boot Attacks on Bliss. In *International Conference on Cryptology and Information Security in Latin America*. Springer, 40–61.
[68] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. 2016. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In *USENIX Security*.
[69] Yuval Yarom and Katrina Falkner. 2014. {FLUSH+ RELOAD}: A High Resolution, Low Noise, L3 Cache {Side-Channel} Attack. In *23rd USENIX security symposium (USENIX security 14)*. 719–732.

A ADDITIONAL ATTACKS AGAINST OTHER NIST LATTICE KEMS

In this appendix we give additional details concerning how a hypothetical Rowhammer attack against each of Kyber, Saber, NTRU-prime and NTRU might work.

A.1 Attacking Kyber

Similar to Frodo, Kyber is an IND-CCA key-capsulation mechanism constructed as a Fujisaki-Okamoto transform of an IND-CPA encryption scheme based on Module-Learning with Error. We defer to Kyber submission available at [45] for details. The design of Kyber follows the same paradigm as Frodo except that Kyber works with polynomials in the ring $R := \mathbb{Z}[x]/(x^{256}+1)$ and prime modulus $q = 3329$. The decoding error in Kyber is

$$\mathbf{e}^T\mathbf{r} + e_2 + c_v - \mathbf{s}^T(\mathbf{e}_1 + \mathbf{c}_u),$$

where $\mathbf{e}, \mathbf{r}, \mathbf{s}, \mathbf{e}_1$, and $\mathbf{c}_u$ are vectors of polynomials in the ring R, and e_2 and c_v are single polynomials. The correctness condition of Kyber places a bound on the magnitude of each coefficient of the polynomial expression above. Here, $\mathbf{s}$ and $\mathbf{e}$ are the secret key and error, analogous to $\mathbf{S}$ and $\mathbf{E}$ in FrodoPKE. The vectors of polynomials $\mathbf{r}$ and $\mathbf{e}_1$ and the polynomial e_2 are sampled during encryption. The terms $\mathbf{c}_u$ and c_v correspond to additional decoding errors introduced by Kyber's compression and decompression of the ciphertext. Compression encodes an element of $\mathbb{Z}_q$ using fewer than $\log_2(q)$ bits and decompression extracts an element of $\mathbb{Z}_q$ from fewer than $\log_2(q)$ bits.

We note that each polynomial $p \in R$ defines a module endomorphism of R as multiplication by p, which has a matrix representation given a fixed basis of R (e.g. the standard basis $\{1, x, x^2, \dots\}$). Therefore, the decoding error above could be read as a matrix expression similar to Frodo decoding error. So our framework applies. In a similar fashion to our attack on Frodo, row-hammering could potentially induce a change in the distribution of $\mathbf{e}$ or $\mathbf{s}$. Combining with a malicious distribution of $\mathbf{r}$ or $\mathbf{e}_1$, the adversary could then obtain an abnormally large decryption rate that could lead to a successful (partial) key recovery.

A.2 Attacking Saber

Saber is based on Module Learning with Rounding (M-LWR). The main differences between Saber and FrodoKEM are:

- The matrix $\mathbf{A} \in \mathbb{Z}_q^{n\ell \times n\ell}$ is structured so that it can be represented as an $\ell \times \ell$ matrix of polynomials of degree $n-1$ in $\mathbb{Z}[x]_q/(x^n+1)$ for M-LWR.
- Rather than adding error vector $\mathbf{e}$, sampled explicitly from some distribution, to $\mathbf{As}$ to produce $\mathbf{b}$, $\mathbf{b}$ is generated by rounding $\mathbf{As}$.

As a result, a Rowhammer adversary can only hammer $\mathbf{s}$ or $\mathbf{b}$. In the reference implementation of Saber, there are a number of calls to SHAKE-128 that may be used for performance degradation similar to what we did with FrodoKEM. While these calls are prior to the generation of $\mathbf{s}$ or $\mathbf{b}$, they can effectively be used to Rowhammer $\mathbf{b}$, since when $\mathbf{b}$ is generated, partial sums are added to a 0 vector which is initialized before the SHAKE-128 calls. We therefore expect the 0 vector can be Rowhammered resulting in a constant value being added to certain coefficients of $\mathbf{b}$. We did not identify a similarly promising strategy for rowhammering $\mathbf{s}$.

A.3 Attacking NTRU-LPRime

NTRU-LPRime is similar in construction to FrodoKEM, Kyber and Saber, but based on Ring Learning With Rounding (R-LWR). It uses the polynomial ring $\mathbb{Z}[x]/(x^p - x - 1)$ where p is an inert prime. In NTRU-LPRime's notation, the polynomial $\mathbf{G}$ is analogous to $\mathbf{A}$ in FrodoKEM, the polynomial $\mathbf{a}$ is analogous to FrodoKEM's $\mathbf{S}$. Like Saber, NTRU-LPRime uses rounding instead of an explicit error analogous to FrodoKEM's $\mathbf{E}$. In NTRU-LPRime's notation $\mathbf{A} = \mathrm{Round}_{\mathbf{aG}}$ is analogous to FrodoKEM's B.

The reference implementation of NTRU-LPRime uses AES instead of SHAKE. It seems harder to do performance degradation on AES, since AES has hardware support. If NTRU-LPRime is implemented with other primitives that are more susceptible to performance degradation or if there is a sufficient performance degradation against AES (in hardware), it appears possible to mount a similar decryption failure attack in a very straightforward way. In principle, a or A (in NTRU-LPRime's notation) are possible targets for Rowhammering.

However, in NTRU-LPRime's reference implementation, a is stored in the private key in a compressed way, which means that any large coefficients in a would create a mismatch between the public and private key and cause decryption to fail with overwhelming probability. Also, in the sampling procedure for a, the calls to AES precede the generation of the coefficients which would be the target for Rowhammering. (A similar situation occurs with the polynomial

f in Streamlined NTRU Prime.) Attacking A looks a little more promising. Rowhammering must occur before the hash of a slightly compressed encoding of A as part of the private key. An AES-based random number generator is called in this window, but it only generates 256 bits of output which might not be a large enough target for performance degradation.

A.4 Attacking NTRU

The design of NTRU is significantly different from that of Frodo, Kyber, and Saber. However, for the purpose of our presentation, it suffices to only consider the correctness condition of NTRU. The correctness condition of NTRU requires that each coefficient of the following polynomial is in the range $[-q/2, q/2)$:

$$3 \cdot \mathbf{r} \cdot \mathbf{g} + \mathbf{f} \cdot \texttt{Lift}(\mathbf{m})) \bmod (x^n - 1), \tag{4}$$

where each term is a polynomial having degree at most $n-2$. Here, $(\mathbf{f}, \mathbf{g})$ is the secret sampled during key generation and $(\mathbf{r}, \mathbf{m})$ is sampled during encapsulation. Decapsulation, if successful, recovers $(\mathbf{r}, \mathbf{m})$ and outputs the shared session key $H(\mathbf{r}, \mathbf{m})$. `Lift` is an injection that maps each $\mathbf{m}$ to a polynomial in $\mathbb{Z}[x]$ such that

$$\texttt{Lift}(\mathbf{m}) \bmod (3, (x^n - 1)/(x - 1)) = \mathbf{m}.$$

Again, we may consider (4) as a matrix expression and apply our framework. We could potentially row-hammer $(\mathbf{f}, \mathbf{g})$ and maliciously select $(\mathbf{r}, \mathbf{m})$ to try to induce larger decryption failure rate. However, difficulty arises. Algebraic operations in NTRU switch between different polynomial moduli, $x^n - 1$, $x - 1$, and $(x^n - 1)/(x - 1)$. Row-hammering must ensure the algebraic constraint that $\mathbf{g} \equiv 0 \pmod{(q, x - 1)}$ and $\mathbf{f}$ is invertible $\bmod (q, (x^n - 1)/(x - 1))$. Otherwise, the result of key generation will cause high failure probability over all ciphertexts and is unusable. On the other hand, because $\mathbf{r}$ and $\mathbf{m}$ can be selected arbitrarily (subject to some algebraic constraints), we may not need to compute many hashes as in Frodo to find a potentially good ciphertext to query for decryption.

A.5 Attacking Streamlined NTRU Prime

Streamlined NTRU Prime has much similarity in its design to NTRU but uses a different polynomial ring $\mathbb{Z}[x]/(x^p - x - 1)$ where p is a prime. Key generation of streamlined NTRU Prime samples two secret polynomials f and g with ternary coefficients, i.e. coefficients in $\{-1, 0, 1\}$. Encapsulation samples a random polynomial r with ternary coefficients such that exactly w coefficients are non-zero, where w is a parameter of the scheme. For correctness, streamlined NTRU Prime requires that each coefficient of the following polynomial is in the range $(-q/2, q/2)$:

$$g \cdot r + 3 \cdot f \cdot e \bmod (x^p - x - 1),$$

where e is a polynomial with ternary coefficients corresponding to error introduced by a certain rounding operation. Decapsulation, if successful, recovers r and outputs the hash of r (concatenated with other public inputs) as shared session key. If row-hammering induces the polynomial g or f to have malformed coefficients, the adversary could then potentially mount a decryption failure attack with knowledge of many tuples (r, e) that cause the expression above to exceed the threshold.

The reference implementation of streamlined NTRU Prime uses AES instead of SHAKE, and it seems harder to do performance degradation on AES. If Streamlined NTRU Prime is implemented with other primitives that are more susceptible to performance degradation (or if there is a sufficient performance degradation against AES), it appears theoretically possible to mount a similar decryption failure attack to our attack on Frodo. Moreover, similar to the case of NTRU, since the polynomial r can selected arbitrarily without any call to a random oracle, generating a candidate failing ciphertext may require much less computation. The Rowhammering phase of such an attack would need to target g rather than f, because f is stored in the secret key in a compressed form that assumes all the coefficients are small enough to be stored in 2 bits. Also, in the sampling procedure for f, unlike the sampling procedure for g, the calls to AES precede the generation of the coefficients which would be the target for Rowhammering.