On the Feasibility of Single-Trace Attacks on the Gaussian Sampler using a CDT

Soundes Marzougui[1], Ievgen Kabin[2], Juliane Krämer[3], Thomas Aulbach[3], and Jean-Pierre Seifert[1,4]

[1] Technische Universität Berlin, Germany
`soundes.marzougui@tu-berlin.de`
`Jean-Pierre.Seifert@external.telekom.de`
[2] IHP - Leibniz-Institut für innovative Mikroelektronik
`kabin@ihp-microelectronics.com`
[3] Universität Regensburg, Regensburg, Germany
{`thomas.aulbach, juliane.kraemer`}`@ur.de`
[4] Fraunhofer Institute for Secure Information Technology, Germany

Abstract. We present a single-trace attack against lattice-based KEMs using the cumulative distribution table for Gaussian sampling and execute it in a real-world environment. Our analysis takes a single power trace of the decapsulation algorithm as input and exploits leakage of the Gaussian sampling subroutine to reveal the session key. We investigated the feasibility of the attack on different boards and proved that the power consumption traces become less informative with higher clock frequencies. Therefore, we introduce a machine-learning denoising technique, which enhances the accuracy of our attack and leverages its success rate to 100%.
We accomplish the attack on FrodoKEM, a lattice-based KEM and third-round alternate candidate. We execute it on a Cortex-M4 board equipped with an STM32F4 micro-controller clocked at different frequencies.

Keywords: FrodoKEM, Gaussian sampler, Machine-Learning, Post-quantum cryptography, Power analysis, Side-channel analysis

1 Introduction

Key encapsulation mechanisms (KEM) are widely adopted in Internet protocols to establish a secure communication between two parties through the encryption of the exchanged messages. Classical KEMs - relying on the intractability of factorization or discrete-logarithm problems for large numbers - are considered to be threatened by attacks with quantum computers [31] in the near future [21]. To address the potential threats from quantum attacks, the National Institute of Standards and Technology (NIST) initiated a standardization process for post-quantum schemes in 2016, i.e., for cryptographic schemes which are assumed to be resistant towards attacks with quantum computers [23]. Post-quantum schemes can be categorized in five different families. Of the five, lattice-based cryptography has received significant research traction in recent years. Lattice-based cryptography consists of schemes whose security can be reduced to the hardness of lattice problems, for instance, the shortest vector problem (SVP), the closest vector problem (CVP), and the learning with errors (LWE) problem. Lattice-based schemes have increasingly attracted attention owing to their balanced performance in terms of sizes and speed and have been studied for real-world deployment [8, 18, 24, 34]. In mid-2022, the NIST standardization process reached the end of the third round and both signature schemes and KEMs have been chosen for standardization.

FrodoKEM is a well-known lattice-based key encapsulation mechanism [4]. It was a third-round NIST alternate candidate and recommended by German Federal Office for Information Security (BSI) [9] and the Netherlands National Communications Security Agency (NBV) [1] for achieving quantum-safe communication. Moreover, FrodoKEM was optimized and included in different libraries such as pqm4 [12] and liboqs [33].

Lattice-based KEMs in general and FrodoKEM in specific are considered to be secure against quantum attacks [31]. However, their practical implementations succumb to side-channel analysis where an adversary has access to the victim's device. Regarding this, one might ask whether it will be easier for an attacker with access to a victim's device to directly extract the session key instead of performing a tedious side-channel analysis. For instance, an employee with the knowledge of the device's technical details can easily extract the secret key. Hence, high-end data protection has become a common place standard for every major business entity. Moreover, the session keys generated during cryptographic processes should be secured even against the manufacturer itself, thus deeming such processes 'maker-proof'. Not just that, these session keys are protected by being stored in trusted physical modules [28,35] which are tamper- and intrusion-resistant, highly-trusted, and meet security standards and regulations used, e.g., in banking systems.

Given that, extracting the session key directly from the device is challenging. Hence, most attacks target a vulnerable routine, extract sensitive information via a side-channel analysis, and build the long-term secret key. With the knowledge of the long-term secret key, the session key can be easily calculated by decrypting the exchanged ciphertext.

For example, an attacker having access to a device can retrieve the secret key by observing the power consumption, the drop of voltage, or timing variation.

Related Work In [26], Ravi et al. identified vulnerabilities in the decapsulation procedure, exploited them to gain information about the decrypted messages, and recovered the long-term secret key within multiple side-channel information. Another attack was proposed by Aysu et al. [3]. The attack targeted the matrix and polynomial schoolbook multiplication used in these protocols. The crux of their attack was to apply a horizontal attack that makes hypotheses on several intermediate values within a single execution, all relating to the same long-term secret, and combine their correlations for estimating the secret key. Despite the fact that their attack needs a single trace, its success highly depends on the accuracy of the used triggering techniques.

In [32], Bo-Yeon Sim et al. analyzed the message encoding operation in the encapsulation phase of different lattice-based KEMs, i.e., CRYSTALS-KYBER, SABER, and FrodoKEM, and obtained the session key with a single power consumption trace [32]. Their experiments show that the success rate of the attack for FrodoKEM can be low to 79%.

In [14], Kim et al. proposed a single trace attack against KEMs employing the cumulative distribution table (CDT) for the error vector sampling. The CDT sampler outputs a sample by sampling first a random value and iterating through the CDT until the entry corresponding to the uniform random value is found. Unlike most of attacks against KEMs in the literature, their attack reveals the session key directly, without the need to obtain the long-term secret key. This attack is convenient tract case ephemeral keys are used and can also be applied to lattice-based encryption algorithms that use CDT for error sampling. The purpose of their attack is to retrieve the sampled error and reduce each LWE instance to a linear relation with the secret information.

In this paper, we investigate the practicability of the attack proposed in [14] in real-world circumstances and provide a proof-of-concept implementation. For that, we use different boards running on different frequencies. We perform our side-channel analysis and show that as compared to the previous work of [14] reading out the samples from a single power consumption trace of the FrodoKEM decapsulation is not feasible. In fact, the measurement gets noisier and less informative with higher frequency and different setup. As a solution, we employ an offline-trained machine-learning classifier on a device similar to the victim's, intended to predict the Gaussian samples during the attack phase. Moreover, we discuss the possible countermeasures to our attack based on literature work.

Contribution In this paper, we investigate the feasibility of single-trace attacks against KEMs using Gaussian sampling and present a full-key recovery for FrodoKEM. We confirm the practicality of our full-key recovery attack targeting the NIST reference implementation and the optimized code of the pqm4 library for FrodoKEM when executed on a 32-bit Arm Cortex-M4 using machine-learning filtering techniques. We prove that our attack is robust to real-world conditions, such as noisy power measurements and high frequencies. The main contributions of the paper are summarized as follows:

- Evaluation of the feasibility of a single-trace attack against FrodoKEM. Our experimental setup relies on different realistic conditions: different clock frequencies and different target boards.
- Proof-of-concept implementation of a single-trace attack against FrodoKEM.
- Deployment of machine-learning tools to retrieve the sensitive information in case of noisy and/or less informative measurements.
- Discussion of a possible countermeasure implementation of Gaussian sampling as suggested in the literature.

Organization: The remainder of this paper is organized in six sections. In Section 2, we give preliminaries on the FrodoKEM scheme and the Gaussian sampling. Then, in Section 3, we present our experimental setup and the targeted implementations. In Section 4, we present a simple power analysis on the Gaussian sampling routine. Subsequently, we present a detailed mathematical description of our single trace power analysis attack in Section 5 and focus on the influence of the introduced noise on the feasibility of our attack. Likewise, in Section 6, we describe our machine-learning filtering techniques. We conclude the paper with Section 7 where we discuss the possible countermeasure and call attention to the urgent need of further protection against single trace attacks targeting the Gaussian sampler.

2 Background

2.1 Lattices

A *lattice* Λ is a discrete subgroup of $\mathbb{R}^n$. Given $m \leq n$ linearly independent vectors $\boldsymbol{b_1}, ..., \boldsymbol{b_m} \in \mathbb{R}^n$, the lattice $\Lambda(\boldsymbol{b_1}, ..., \boldsymbol{b_m})$ is the set of all integer linear combinations of the $\boldsymbol{b_i}$'s, i.e.,

$$\Lambda(\boldsymbol{b_1}, ..., \boldsymbol{b_m}) = \Big\{ \sum_{i=1}^{m} x_i \boldsymbol{b_i} \;\Big|\; x_i \in \mathbb{Z} \Big\},$$

where $\boldsymbol{b_1}, ..., \boldsymbol{b_m}$ form a *basis* of Λ and m is the *rank*. In this paper, we consider full-rank lattices, i.e., with $m = n$. An *integer lattice* is a lattice for which the basis vectors are in $\mathbb{Z}^n$. Usually, we consider elements modulo q, i.e., the basis vectors and coefficients are taken from $\mathbb{Z}_q$.

2.2 Learning with Errors

The Learning with Errors problem (LWE), which is a generalization of the classic Learning Parities with Noise problem, was introduced by Regev [27]. We explain in the following the Learning with Errors problem.

Definition 1. *Let n, q be positive integers, and let χ be a distribution over $\mathbb{Z}$. For $\boldsymbol{s} \in \mathbb{Z}_q^n$, the LWE distribution $A_{s,\chi}$ is the distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing $\boldsymbol{a} \in \mathbb{Z}_q^n$ uniformly at random and an integer error $e \in \mathbb{Z}$ from χ. The distribution outputs the pair $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e \bmod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.*

There are two important computational LWE problems:

- The *search problem* is to recover the secret $\boldsymbol{s} \in \mathbb{Z}_q^n$ given a certain number of samples drawn from the LWE distribution $A_{\boldsymbol{s},\chi}$.
- The *decision problem* is to distinguish a certain number of samples drawn from the LWE distribution from uniformly random samples.

2.3 Gaussian Sampler and CDTs

Several lattice-based schemes use the discrete Gaussian distribution as error distribution χ in Definition 1. A centered discrete Gaussian distribution is used in LWE to ensure that the noise added to the ciphertext is truly random and unbiased, providing a secure and efficient encryption scheme. The discrete Gaussian distribution over a lattice Λ is defined as

$$D_{\Lambda,\sigma}(x) = \frac{\rho_\sigma(x)}{\sum_{y\in\Lambda}\rho_\sigma(y)}, \tag{1}$$

where

$$\rho_\sigma(x) = e^{\frac{-x^2}{2\sigma^2}} \tag{2}$$

represents the continuous Gaussian function. When sampling from the positive integers, we simply write D_σ^+, which is defined accordingly by

$$D_\sigma^+(x) = \frac{\rho_\sigma(x)}{\sum_{y=0}^{+\infty}\rho_\sigma(y)} \tag{3}$$

There are different generic ways to sample from a discrete Gaussian distribution. One of the approaches employs the CDT for sampling as in Algorithm 1. First, the CDT ψ is precomputed using the cumulative distribution function of D_σ^+. The idea is that the sampler returns the index i of the table ψ, such that $\psi[i] < x \leq \psi[i+1]$, where x is generated uniformly from the interval that is covered by the table. The parameter τ denotes the tail-cut and is chosen such that the probability for drawing from outside the interval is negligible.

Algorithm 1 Gaussian Sampler using CDT

```
Require: CDT $\psi$ of length $l$, following a distribution $D_\sigma^+$, and having a tailcut $\tau$
Ensure: Sampled value $S$ following the targeted distribution $D_\sigma$
1: $S \leftarrow 0$
2: $rnd \leftarrow [0,\tau\sigma) \cup \mathbb{Z}$ uniformly at random
3: $sign \leftarrow [0,1] \cup \mathbb{Z}$ uniformly at random
4: for $(i = 0\ ;\ i < l - 1;\ i++)$ do
5:     $S \mathrel{+}= (\psi[i] - rnd) >> 15$
6: end for
7: $S \leftarrow ((-sign) \wedge S) + sign$
8: return $S$
```

2.4 Description of FrodoKEM

FrodoKEM is a lattice-based KEM with security based on the standard LWE problem (i.e., not the ring version of the LWE problem). It is a conservative design with security proofs. KEM's are defined as a triple of algorithms (KeyGen, Encaps, Decaps). KeyGen is the algorithm responsible for public and secret key generation. The encapsulation algorithm Encaps generates a random key $\boldsymbol{k}$ and encrypts it using the public key pk to create a ciphertext c and derive a shared secret $\boldsymbol{ss}$. The decapsulation algorithm Decaps decrypts c using the secret key sk and returns the derived session key $\boldsymbol{ss}$, or a random output in case the re-encrypted ciphertext does not fully match the previous output of the encapsulation algorithm.

We describe the decapsulation of FrodoKEM as it is the target of our attack, see Algorithm 2. We introduce the following parameters:

- $n, \bar{m}, \bar{n}$ integer matrix dimensions with $n \equiv 0 \pmod 8$
- B is the number of the bits encoded in each matrix entry
- $len_{seed_{\boldsymbol{A}}}$ the bit length of seeds used for pseudorandom matrix generation
- $len_{seed_{\boldsymbol{SE}}}$ the bit length of seeds used for pseudorandom bit generation for error sampling
- Gen pseudorandom matrix generation algorithm
- T_χ distribution table for sampling
- $len_{\boldsymbol{s}}$ the length of the bit vector $\boldsymbol{s}$ used for pseudorandom shared secret generation in the event of decapsulation failure
- $len_{\boldsymbol{z}}$ the bit length of seeds used for pseudorandom generation of $seed_{\boldsymbol{SE}}$
- $len_{\boldsymbol{k}}$ the bit length of intermediate shared secret $\boldsymbol{k}$
- $len_{\boldsymbol{pkh}}$ the bit length of the hash of the public key
- $len_{\boldsymbol{ss}}$ the bit length of shared secret $\boldsymbol{ss}$

The decapsulation starts with the calculation of the matrix $\boldsymbol{M}$. When simplifying $\boldsymbol{M}$, we can write it as

$$\boldsymbol{M} = Encode(\mu') + \boldsymbol{S'E} - \boldsymbol{E'S} + \boldsymbol{E''},$$

where $\boldsymbol{S}$, $\boldsymbol{S'}$, $\boldsymbol{E}$, $\boldsymbol{E'}$, and $\boldsymbol{E''}$ have small entries. Therefore, $\boldsymbol{S'E} - \boldsymbol{E'S} + \boldsymbol{E''}$ will also result in a matrix with small entries, regarded as noise. The $Decode$ (line 4, Algorithm 2) removes this noise and returns the seed μ'. The decapsulation then continues by doing a reencryption and comparing the ciphertexts. If the ciphertexts (line 16, Algorithm 2) are equal, the correct shared key $\boldsymbol{ss}$ is returned.

Algorithm 2 FrodoKEM Key Decapsulation according to [23]

```
Require: Ciphertext $\boldsymbol{c_1}\|\boldsymbol{c_2} \in \{0,1\}^{(\bar{m}.n+\bar{m}.\bar{n})D}$, and secret key $sk' = (\boldsymbol{s}\|seed_{\boldsymbol{A}}\|\boldsymbol{b}, \boldsymbol{S}^T, \boldsymbol{pkh}) \in \{0,1\}^{len_s+len_{seed_{\boldsymbol{A}}}+D.n.\bar{n}} \times \mathbb{Z}_q^{\bar{n}\times n} \times \{0,1\}^{len_{pkh}}$
Ensure: Shared secret key $\boldsymbol{ss} \in \{0,1\}^{len_{\boldsymbol{ss}}}$
 1: $\boldsymbol{B'} \leftarrow$ Unpack($c_1$)
 2: $\boldsymbol{C} \leftarrow$ Unpack($c_2$)
 3: Compute $\boldsymbol{M} \leftarrow \boldsymbol{C} - \boldsymbol{B'S}$
 4: Compute $\mu' \leftarrow$ Decode($\boldsymbol{M}$)
 5: Parse $pk \leftarrow seed_{\boldsymbol{A}}\|\boldsymbol{b}$
 6: Generate pseudorandom values
    $seed_{\boldsymbol{S'E'}}\|\boldsymbol{k'} \leftarrow$ SHAKE($\boldsymbol{pkh}\|\mu', len_{seed_{\boldsymbol{SE}}} + len_{\boldsymbol{k}}$)
 7: Generate pseudorandom bit string
    $(r^{(0)}, r^{(1)}, \ldots, r^{(2\bar{m}n+\bar{m}\bar{n}-1)}) \leftarrow$ SHAKE $(0x96\|seed_{\boldsymbol{SE'}}, (2\bar{m}n + \bar{m}\bar{n}).len_{\chi})$
 8: Sample error matrix $\boldsymbol{S'} \leftarrow$ SampleMatrix($r^{(0)}, r^{(1)}, \ldots, r^{(\bar{m}n-1)}, \bar{m}, n, T_{\chi}$)
 9: Sample error matrix $\boldsymbol{E'} \leftarrow$ SampleMatrix($r^{(\bar{m}n)}, r^{(\bar{m}n+1)}, \ldots, r^{(2\bar{m}n-1)}, \bar{m}, n, T_{\chi}$)
10: Generate $\boldsymbol{A} \leftarrow$ Gen($seed_{\boldsymbol{A}}$)
11: Compute $\boldsymbol{B''} \leftarrow \boldsymbol{S'A} + \boldsymbol{E'}$
12: Sample error matrix $\boldsymbol{E''} \leftarrow$ SampleMatrix($r^{(2\bar{m}n)}, \ldots, r^{(2\bar{m}n+\bar{m}\bar{n}-1)}, \bar{m}, \bar{n}, T_{\chi}$)
13: $\boldsymbol{B} \leftarrow$ Unpack($\boldsymbol{b}, n, \bar{n}$)
14: Compute $\boldsymbol{V} \leftarrow \boldsymbol{S'B} + \boldsymbol{E''}$
15: Compute $\boldsymbol{C'} \leftarrow \boldsymbol{V}+$ Encode($\mu'$)
16: if $\boldsymbol{B'}\|\boldsymbol{C} = \boldsymbol{B''}\|\boldsymbol{C'}$ then
17:     return shared secret $\boldsymbol{ss} \leftarrow$ SHAKE $(\boldsymbol{c_1}\|\boldsymbol{c_2}\|\boldsymbol{k'}, len_{\boldsymbol{ss}})$
18: else
19:     return shared secret $\boldsymbol{ss} \leftarrow$ SHAKE $(\boldsymbol{c_1}\|\boldsymbol{c_2}\|\boldsymbol{s}, len_{\boldsymbol{ss}})$
20: end if
```

3 Experimental Setup

In this section, we present the experimental setup used for our side-channel analysis. Our attack targets the implementations taken from the open-source pqm4 library [12], running on the ARM Cortex-M4 and Harvard micro-controllers [5].

3.1 Implementations of FrodoKEM

The FrodoKEM source code [2] was provided as a portable C implementation. The reference implementation of FrodoKEM having the smallest parameter set (frodokem640shake/ frodokem640aes) requires almost a megabyte of RAM (including messages and keys). This is mainly due to placing the entire matrix $\boldsymbol{A}$ in RAM. For larger parameter sets, more memory will be required. Therefore, the reference implementations are not suitable for the target platforms considered for this attack (and less interesting as side-channel target anyway). The optimized

[5] The implementation of our attack can be found at `https://github.com/Soundes-M/Soundes-M-FrodoKEMSingleTrace-/settings`

implementations provided by pqm4 reduce the memory consumption, however, the memory footprint remains large, and only the variant of NIST security level I (frodokem640shake) fits on STM32F[6] target platforms consuming 117 KB of RAM (including messages and keys). The larger parameter sets of FrodoKEM consume between 181 KB (frodokem976shake) and 298 KB (frodokem1344aes). This is due to the fact that the implementations of the AES parameter sets of FrodoKEM use more memory than their SHAKE counterparts, and, thus, exceed our memory limits by far [13]. Additionally, pqm4 [12] includes M4-optimized assembly implementations for frodokem640shake and frodokem640aes by [5]. It speeds up the polynomial multiplication and decreases the stack memory consumption [5]. In that context, we mount our attack against the optimized [12] implementation of frodokem640shake.

3.2 Experimental Workbench

To record the traces for the attack, we used two different target boards,i.e., 8-bit Harvard and 32-bit Cortex, mounted on a ChipWhisperer Lite CW308 UFO as in Figure 1. The ChipWhisperer is equipped with an analog-to-digital (ADC) which converts the voltage input to a digital number representing the magnitude of the voltage.

During recording, the ChipWhisperer and the micro-controller are synchronized. The sampling rate of the analog-to-digital converter (ADC) was set to 4 samples/cycle with 10-bit resolution. We used a Python script running on the PC to collect and store all relevant traces. We also used a high pass filter in the first experiment to remove the low-frequency noise [10]. For higher frequency, we used an external crystal Quarz oscillator.

The reason behind picking the two architectures 8-bit Harvard and 32-bit Cortex is that we wanted to figure out the feasibility of the single trace attack claimed in [14]. The findings in [14] need to be interpreted with cautions. In fact, in [14], the authors used an 8-bit Harvard board equipped with an XMega micro-controller which is especially common in educational embedded applications. In contrast, in real world, Cortex-M boards have been embedded in tens of billions of consumer devices.

Target Boards Setup Our setup is composed of a CW308 UFO Board which is a board suitable for attacking different sorts of embedded targets. The UFO board has three 20-pin female headers into which the target board fits. They provide both electrical and mechanical connections for the board. The pin 1 in the right of the UFO board (Figure 1) corresponds to the low-side shunt connection connected to the SMA cable, which is a coaxial cable responsible for transmitting the signal from the target board to the ChipWhisperer. The power consumption measurements are obtained by measuring the voltage drop across the shunt resistor.

[6] We use in our experiments the STM32F4 target board which has 1 MB of Flash memory and 192 KB of RAM

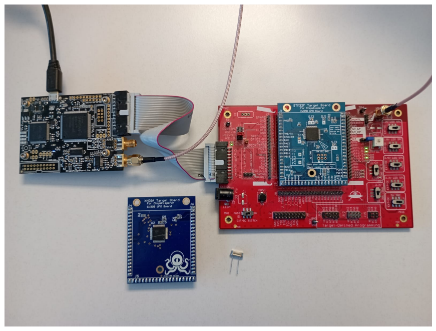


Figure 1: Experimental workbench used for our side-channel analysis; it contains two target boards which can be mounted on the UFO board (red), a ChipWhisperer, an external clock oscillator, and a USB cable.

Clock Frequency Setup The CW308 has a crystal oscillator driver, which allows the attacker to drive the victim board through the use of an external crystal. In other words, it is possible to generate any frequency by simply putting an appropriate crystal (as shown in Figure 1) into the socket. This step needs to be followed by adjusting of the baudrate, and routing the specific crystal oscillator to the victim external clock interface (CLKIN) using jumper J3.

4 Simple Side-Channel Analysis

4.1 Threat Model

Our threat model follows the power side-channel model of [19,20,22]. We assume that an attacker has physical access to the victim's device and is equipped with a reasonable measurement setup that can synchronize the sampling rate within the CPU clock period of the victim's device such as the experimental setup described in Section 3. The adversary in our model can record the power consumption measurement of the key decapsulation (Algorithm 2).

Although we specify the points for our experiments where the execution of the leaking routine (that is, the Gaussian sampling) starts in the power consumption using triggering techniques, we note that for other devices, the attacker might need engineering aspects of locating these cryptographic routine sub-traces among the whole trace as explained in [6, 11, 37].

We indicate that our attack is a passive attack; by revealing the session key, the adversary can also see the encrypted messages over a public channel and has the public key of the victim which is stored in his certificate. However, the adversary is not able to modify, drop, replay, or inject messages on the public channel, nor use the retrieved session key to interact with the other party. Basically, the attacker will be able only to decrypt the exchanged encrypted messages between the two parties using the extracted session key.

4.2 Single-Trace Attack on the Gaussian Sampler

In the FrodoKEM reference and optimized implementation [2,12], the values are expressed in 16-bit integers, and nine bits are used for sampling. When the value is negative and expressed in two's complements, its significant bit is 1.

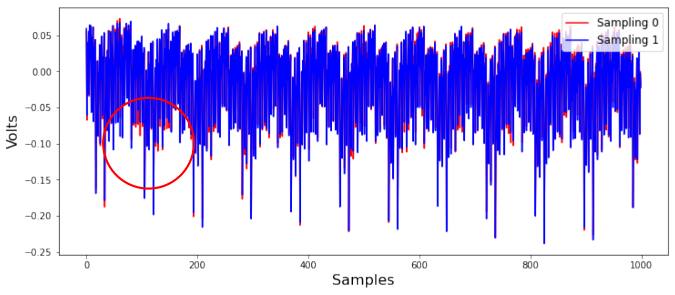
Listing 1.1: Gaussian Sampler Implementation in the reference and optimized implementation of FrodoKEM [2,12]

```
 1 void frodo_sample_n(uint16_t *s, const size_t n) {
 2 unsigned int i, j;
 3     for (i = 0; i < n; ++i) {
 4       uint16_t sample = 0;
 5       uint16_t prnd = s[i] >> 1;
 6       uint16_t sign = s[i] & 0x1;
 7      for (j = 0; j < (unsigned int)(CDF_TABLE_LEN - 1)
 8       ; j++) {
 9         sample += (uint16_t)(CDF_TABLE[j] - prnd) >> 15;
10       }
11     s[i] = ((-sign) ^ sample) + sign;
12     }
13 }
```

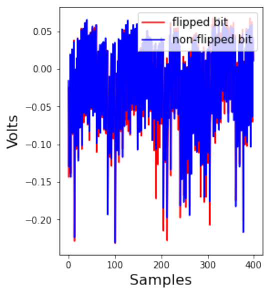
Hence, if the subtraction in line 9, Listing 1.1 yields a negative number, its most significant bit is 1. When this number, expressed in 16 bits, is shifted to the right, one is added to the value *sample*. However, if the subtraction outputs a positive number, its most significant bit is 0. Owing to this, the value of *sample* will not be incremented. By examining the power consumption trace during the iteration through the CDT (Figure 2 (a)), we could distinguish between the addition of *zero* and *one*.

After the iteration through the CDT, the sampler calculates a positive integer called *sample*. Then, a bit-flipping operation is applied (line 11 in Listing 1.1). If the sign bit is flipped and yields zero, then the sample's sign remains the same. Else, the sign is flipped. This can be observed clearly in Figure 2 (b).

Power Consumption Traces with Different Boards We acknowledge that the observations in Figure 2 were already investigated by Kim and Hong in [14]. However, they are not always detectable, especially when taking the measurements on boards with different architectures, such as Cortex-M4. We exemplify this through the power consumption trace of one iteration through the CDT on a Cortex-M4 equipped with an STM32F4 Micro-controller in Figure 3. The results presented in [14] do not apply to our setup. As in Figure 3 (a), one cannot differentiate with the naked eye between the two colors corresponding to the addition of zero and one. Surprisingly, the bit flipping is still vulnerable in this setup as in Figure 3 (b).



(a)



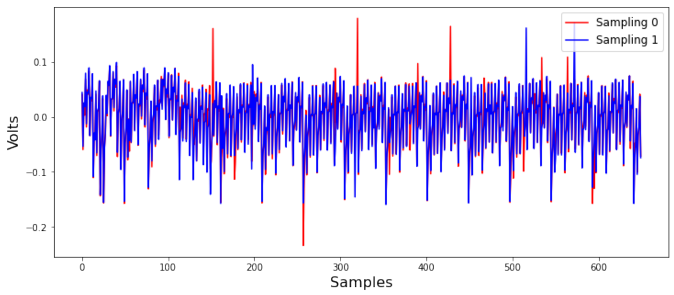
(b)

Figure 2: Figure (a) shows overlapped power consumption measurements during the execution of line 7 - 10 of Listing 1.1, while Figure (b) shows overlapped power consumption measurements during the sign bit flipping operation (line 11, Listing 1.1). Both measurements are taken on an 8-bit Harvard board equipped with an XMega micro-controller; the red color corresponds to the sampling of the value 0 in Figure (a) and a flipped bit in Figure (b), while the blue color corresponds to the sampling of the value 1 in Figure (a) and a non-flipped bit in Figure (b)

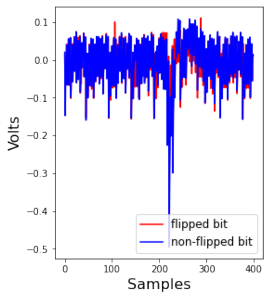
Explanation For each target, the C code gets compiled and outputs a binary. The binaries corresponding to the compilation of the C code for different targets are identical, however, they are meant to be executed on targets that are in reality each very different from one another. Each target is a distinct collection of millions of transistors. Hence, this explains the differences in the power consumption traces, which are the measurement of the power consumed by these millions of transistors. The feasibility of side-channel analysis against one board does not imply necessarily its feasibility on other boards. In the following, we present power consumption measurements on different boards having different frequencies, and we show that the results in [14] do neither apply for all types of boards nor for different clock frequencies.

Power Consumption Traces with Different Frequencies We increased the frequency from 7,327 MHz to 30 MHz and we took the power consumption measurement as in Figure 4 and 5. The red color corresponds to the power consumption traces taken at a frequency of 7,327 MHz, while the blue color indicates those taken at 30 MHz. When the chip is clocked at 7,327 MHz, we notice (as in the randomly zoomed area in Figure 4 and 5) that in every clock cycle there are two high peaks. However, at a higher frequency we notice that only one peak is occurring which minimized the fluctuation of the power consumption measurement as compared to those taken at lower frequency.

Explanation We explain this behaviour based on [17]. There are two peaks (at the low frequency 7,327 MHz). The high peak occurs when the clock edge rises from low to high, smaller peaks take place when the clock signal falls down. At a



(a)



(b)

Figure 3: Figure (a) shows overlapped power consumption measurements during the execution of line 7 - 11 of Listing 1.1, while Figure (b) shows overlapped power consumption measurements during the sign bit flipping operation (line 11, Listing 1.1). Both measurements are taken on on a Cortex-M4 equipped with an STM32F4 micro-controller; the red color corresponds to the sampling of the value 0 in Figure (a) and a flipped bit in Figure (b), while the blue color corresponds the sampling of the value 1 in Figure (a) and a non-flipped bit in Figure (b).

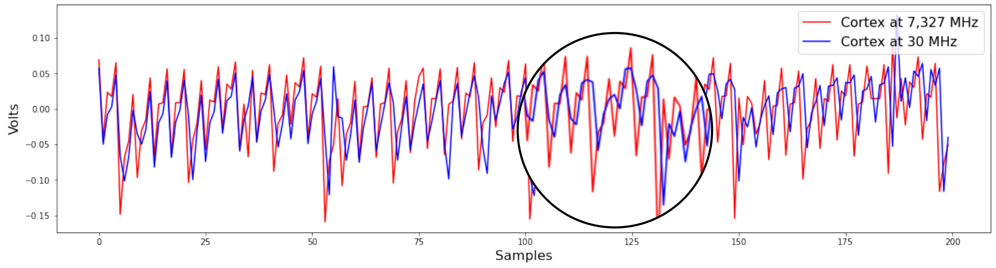


Figure 4: Overlapped Power consumption measurement during the execution of line 9 of Listing 1.1 on an 32-bit Cortex board equipped with an STM32F4 micro-controller for two different clock frequencies

higher clock frequency, the small peak fades or even vanishes as it overlaps with the highest peak. This yields a less informative measurement as the number of peaks of the power consumption traces decreases with higher frequencies. To this end, we came to the conclusion that single-trace attacks against the CDT Gaussian sampler presented in [14] cannot be generalized to different boards and different frequencies.

5 Description of the Attack and Error Tolerance

Once the victim starts executing the decapsulation algorithm, the attacker records its power consumption. Then, the attacker can locate in the full-trace the power consumption subtrace $T_{S'}$ and $T_{E''}$ corresponding to the sampling of $\boldsymbol{S'}$ and $\boldsymbol{E''}$ as in Algorithm 2. The experimental setup is detailed in Section 3.

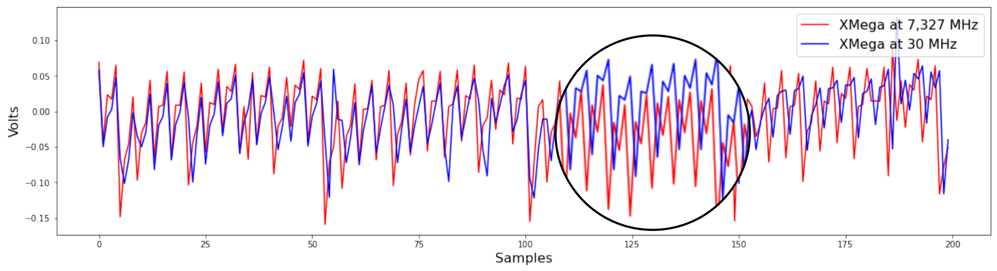


Figure 5: Overlapped Power consumption measurement during the execution of line 9 of Listing 1.1 on an 8-bit Harvard board equipped with an XMega microcontroller for two different clock frequencies

The first subtrace of the $\boldsymbol{S'}$ sampling called $T_{S'}$ is itself composed of 5120 subtraces, each corresponding to one iteration through the CDT. We call these subtraces $T_{S'_i}$, with $0 \leq i < 5120$. The attacker performs the side-channel analysis (described in Section 4) of the collected subtraces to predict the Gaussian samples which are the 5120 entries of the matrix $\boldsymbol{S'}$. We refer to those side-channel analyses by the function *side_channel* (line 4 and 7, Algorithm 3). Similarly, the attacker gets the subtrace $T_{E''}$ corresponding to the sampling of the 64 entries of the matrix $\boldsymbol{E''}$. Again, the attacker analyses each subtrace $T_{E''_j}$ to predict the 64 entries of the matrix $\boldsymbol{E''}$, where $0 \leq j < 64$.

Having the values of $\boldsymbol{S'}$ and $\boldsymbol{E''}$, the attacker computes the matrix $\boldsymbol{V}$ as in the decapsulation (Algorithm 2) through the following equation:

$$\boldsymbol{V} = \boldsymbol{S'}\boldsymbol{B} + \boldsymbol{E''} \tag{4}$$

It is known from Algorithm 2 that:

$$\boldsymbol{C'} = \boldsymbol{V} + Encode(\mu') \tag{5}$$

Then, the attacker obtains the matrix Encode(μ') by plugging Equation 4 in Equation 5 as below.

$$\boldsymbol{C'} = \boldsymbol{S'}\boldsymbol{B} + \boldsymbol{E''} + Encode(\mu') \tag{6}$$

Note that Encode is a function that takes a bit strings of length $l = B \times \bar{m} \times \bar{n}$ as input and encodes it to a matrix of $\bar{m} \times \bar{n}$ entries. We refer to [4] for more details. The Equation 6 gives the encoding of μ'.

$$Encode(\mu') = \boldsymbol{C'} - \boldsymbol{S'}\boldsymbol{B} - \boldsymbol{E''} \tag{7}$$

To obtain μ the attacker applies the *Decode* on the right side of the Equation 7. Note that *Decode* decodes an m-by-n matrix into a bit string of length $B \times m \times n$. This means, it extracts B bits from each entry of the matrix. Hence, μ' can be written as:

$$\mu' = Decode(\boldsymbol{C'} - \boldsymbol{S'}\boldsymbol{B} - \boldsymbol{E''}) \tag{8}$$

The attacker then calculates the session key $\boldsymbol{ss}$. The attack is summarized in Algorithm 3.

Algorithm 3 Single Trace Attack

```
Require: Ciphertext $\boldsymbol{c_1} \| \boldsymbol{c_2} \in \{0,1\}^{(\bar{m}.n+\bar{m}.\bar{n})D}$ and power consumption traces
    $T_{S'} = (T_{S'_0}, \ldots, T_{S'_{\bar{m}n-1}})$ and $T_{E''} = (T_{E'_0}, \ldots, T_{E'_{\bar{m}\bar{n}-1}})$
Ensure: The session key $\boldsymbol{ss}$
 1: $\boldsymbol{B'} \leftarrow$ Unpack$(c_1)$
 2: $\boldsymbol{C} \leftarrow$ Unpack$(c_2)$
 3: for $i \in \{0 \ldots \bar{m}n\}$ do
 4:     $S'_i \leftarrow side\_channel(T_{S'_i})$
 5: end for
 6: for $i \in \{0 \ldots \bar{m}\bar{n}\}$ do
 7:     $E''_i \leftarrow side\_channel(T_{E''_i})$
 8: end for
 9: Compute $\boldsymbol{V} = \boldsymbol{S'B} + \boldsymbol{E''}$
10: Compute $Encode(\mu') = \boldsymbol{C} - \boldsymbol{V}$ and get $\mu'$ by applying $Decode()$
11: Generate pseudorandom values $seed_{\boldsymbol{SE'}} \| \boldsymbol{k'} \leftarrow$ SHAKE$(\boldsymbol{pkh} \| \mu', len_{\boldsymbol{SE}} + len_{\boldsymbol{k}})$
12: shared secret $\boldsymbol{ss} \leftarrow$ SHAKE $(\boldsymbol{c_1} \| \boldsymbol{c_2} \| \boldsymbol{k'}, len_{\boldsymbol{ss}})$
13: return $\boldsymbol{ss}$
```

Noise Tolerance We notice here that the noise on the samples $\boldsymbol{E''}$ does not affect the correctness of the attack because the $Decode()$ function rounds the term $\boldsymbol{C'} - \boldsymbol{S'B} - \boldsymbol{E''}$ to the $(32 - B)$ most significant bits. Hence, the B least significant bits -even when guessed wrong- do not have consequences on the value μ' (B is equal to 2 for the first security level of FrodoKEM).

On the other side, having an error on one coefficient of the matrix $\boldsymbol{S'}$ will propagate when this latter is multiplied by $\boldsymbol{B}$ and will result in different erroneous matrix entries. This error can be located in the first bits of the coefficients of the resulting matrix ($\boldsymbol{S'B}$) which will not be tolerated and result in a wrong value of μ'. We include in the following section, a machine-learning side channel analysis technique that leverage the accuracy of the single trace attack and enhance the attacker capability to retrieve the session key.

6 Machine-Learning Side-Channel Analysis

Side-channel experiments are usually carried out in careful and non-realistic circumstances. For example, the noise is minimized by using noise filtering techniques, or non-commercial prototype boards are targeted. This yields easy-to-analyse power consumption traces. However, in reality the real-world conditions such as noise can prohibit the attack as explained in Section 5. To tackle this problem, we write the power consumption [7] at a specific point of time as the following:

$$P = P_{op} + P_{data} + P_{noise} + P_{const} \tag{9}$$

[7] We mean here the power consumption of the device while running a cryptographic operation.

The four entities P_{op}, P_{data}, P_{noise}, and P_{const} are functions of time. The entities P_{op} and P_{data} are the power consumption depending on the executed operation and the data, respectively. The entity P_{noise} represents the noise added to the power consumption measurement, and P_{const} refers to constant power consumption that occurs independently of the operation and the data.

Power analysis exploits the dependency between the power consumption and the processed operation and data, i.e., P_{op} and P_{data}. In Section 4.2, we figured out two scenarios affecting the ability of an attacker to analyse the power consumption traces. In the first scenario, the noise level is extremely high, i.e., $P_{noise} >> P_{op} + P_{data}$, which prevents the attacker from detecting the leakage. To characterize the added noise P_{noise}, we recorded the power consumption while running the Gaussian sampling with fixed input, i.e., fixed random numbers. This yields the repeated execution of the same instructions with the same input data. We mounted this experiment with 1000 repetitions on each board. From each of these traces we took always the sample with the same index (i.e., index 120) corresponding to the first point of interest (POI) in the power consumption trace. With these points we computed the two histograms in Figure 6 and 7.

For a low frequency, we point out that most of the points are concentrated around zero. The shape of the histogram in Figure 6 indicates that the points in the power traces follow a Gaussian distribution. However, for higher frequency the noise distribution tends to be uniform for a cortex M4 board. For an XMega board, the added noise follows a multidimensional Gaussian distribution.

In the second scenario, the high frequency leads to flattening the power consumption trace, i.e., $P_{op} + P_{data} \sim \textit{constant}$ (observations in Section 4.1 demonstrate that with higher frequencies the power consumption becomes less informative) which can often decrease the capability of an attacker to retrieve the intermediate sensitive data with the naked eye or even with differential power analysis.

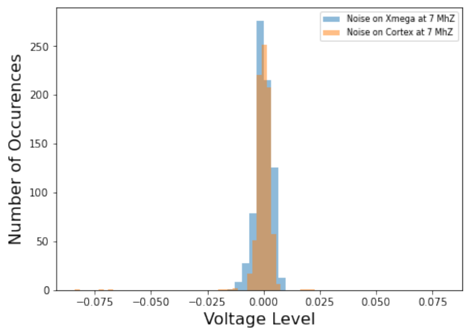


Figure 6: Noise Distribution on POI when the target board is clocked at 7 MHz

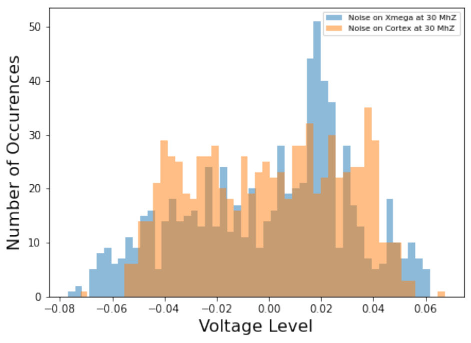


Figure 7: Noise Distribution on POI when the target board is clocked at 30 MHz

Therefore, in both scenarios, a single-trace attack against FrodoKEM as in [14] could not be mounted in real circumstances. To tackle these challenges, one way is to use a spectrum analyser which displays a spectrum of signal amplitudes on different frequencies and determines the noise added to the signal (power consumption of the device while running the cryptographic operation). In view of the expensive price of a spectrum analyzer, other cheaper ways are considered, e.g., template attacks based on a Gaussian assumption [7]. However, template attacks have been subject to much criticism [16]. First, template attacks hold true only if the assumption on the noise distribution is correct. This is not always fulfilled in a real-world scenario. Moreover, they are useful as long as a limited number of points of interest can be identified in leakage traces and contain most of the information. If the number of useless samples in leakage traces increases and/or the size of the profiling set becomes too limited, the template attacks are useless [16]. Instead, machine-learning side-channel analysis is more powerful in this case.

We propose a machine-learning technique composed of two phases: a profiling phase and an attack phase.

6.1 Profiling Phase

To prepare the profiling, we executed the Gaussian sampling process (Algorithm 1) with random input.

The training of our neural network proceeds as follows. First, we prepare traces of the execution of multiple iterations through the CDT during the Gaussian sampling function with random input (lines 7–13, Listing 1.1). The output of each CDT iteration (i.e., the Gaussian sample) is assigned as the label. With the prepared traces and their corresponding labels, we can build a list of examples $(x, y) \in \mathbb{R}^t \times Y$, where $x \in \mathbb{R}^t$ are the power traces acting as the *features*, and $y \in Y$ are the Gaussian samples acting as the *labels*. We assume that x leaks information about y. The list of these noisy examples (x, y) is split into training, validation, and a test set of the Multi-Layer Perceptron (MLP) machine-learning classifier.

We emphasize that the attacker should train a classifier for each board type (i.e., Cortex-M4, Harvard) and frequency. According to [36], ignoring the board's specifics and diversity can easily lead to an overestimation of the classification accuracy.

Tuning the hyper-parameters of our machine-learning model is of particular importance because it influences the accuracy of the trained machine-learning classifier, hence the practicability of our attack. There are often general heuristics or rules of thumb for configuring hyper-parameters. However, a better approach is to objectively search different values for model hyper-parameters and choose a subset that maximizes the prediction accuracy of our classifier. The so-called hyper-parameter optimization is available in the scikit-learn Python machine-learning library [30]. The result of a hyper-parameter optimization is a single set of well-performing hyper-parameters that is used to configure the MLP classifiers.

We refer to each of our MLP classifiers as *Classifier* which is trained on traces labeled by the output samples. We captured 20,000 power consumption traces. We set 18,000 of them for training and testing and 2,000 for validation.

6.2 Attack Phase

Equipped with the trained MLP classifier, the attacker deduces information about the samples from the power traces.

First, she (the attacker) lets the victim device run the decapsulation process. Then, she extracts from the full power trace the parts of the trace corresponding to the Gaussian sampling of the matrices $\boldsymbol{S'}$ and $\boldsymbol{E''}$. Each of those traces, i.e., $\boldsymbol{S'}$ and $\boldsymbol{E''}$ is split itself into snippets corresponding to the sampling of the matrix entries. To this end, the attacker feeds those snippets to the classifier trained in the profiling phase, and obtains a prediction of each entry in the matrices $\boldsymbol{S'}$ and $\boldsymbol{E''}$.

Once the matrices $\boldsymbol{S'}$ and $\boldsymbol{E''}$ are predicted by the attacker, this latter plugs this information in Equation 8 and calculates the value μ' as described in Section 5. Therefore, the session key can be calculated by hashing the value of μ' concatenated with the public known values of the ciphertext, $\boldsymbol{k'}$, and $len_{\boldsymbol{ss}}$ as the following: $\boldsymbol{ss} = \text{SHAKE}\ (\boldsymbol{c_1} \| \boldsymbol{c_2} \| \boldsymbol{k'}, len_{\boldsymbol{ss}})$.

7 Countermeasures and Conclusion

It is important to note that CDT is not the only method of sampling, with binomial sampling being used in modern lattice-based schemes such as Kyber. Our attack may potentially affect the binomial distribution as similar instructions are executed, and we anticipate it to be even more efficient due to the fact that binomial sampling counts the number of positive outcomes in binary experiments, making bitwise operations more susceptible to attack through side-channel analysis. In the following, we discuss the possible countermeasures against the proposed single-trace attack on the CDT sampler.

To avoid the presented single trace attack, the work [38] *might* be a countermeasure. Instead of sampling directly from the target distribution D_σ, they suggest to first start by generating a sample x from the base sampler $D^+_{\sigma_0}$, where $\sigma_0 = \sqrt{1/2 \ln(2)}$.

Then, obtain the value y_u uniformly at random from $[0, \cdots, K-1]$, and compute $z = y_u + Kx$, where $K = \left\lfloor \frac{\sigma}{\sigma_0} + 1 \right\rfloor$ is a constant. Finally, a Bernoulli rejection sampling with acceptance rate

$$p = \exp\left(\frac{-y(y+2Kx)}{2\sigma^2}\right)$$

is applied to ensure that z follows the distribution D^+_σ. In order to obtain also negative samples, one can apply a random sign bit, but has to reject $z = 0$ with probability $1/2$.

This countermeasure *does* protect against the proposed single-trace attack for schemes having relatively high deviation. However, for FrodoKEM the target standard deviation is $\sigma = 2.8$. Therefore, the value of K is small, i.e $K = 4$. An attacker still can get the samples with a precision K, since she is able to reveal the value of x with the introduced single trace attack. She only misses out on the value $y_u \in \{0, \cdots, K-1\}$, when trying to recover the final sample $z = y_u + Kx$.

One of the common countermeasures is masking. In [29] Schneider et. al presented the first protected (masked) binomial sampler claimed to be secure against side-channel adversaries of arbitrary order. First-order masking countermeasures are not enough to protect against ML attacks. Leveraging the masking order is needed but it yields a performance overhead memory and time-wise.The first-order masking technique has been, however, broken by [22] using machine-learning side-channel analysis.

Another countermeasure was suggested by [14], where a look-up table is used for sampling. One samples a 16-bit integer, and uses it as the index of the look-up table and outputs the corresponding table value (or the sample). However, this countermeasure needs to store a big look-up table. For FrodoKEM-640, the Gaussian sample does not exceed one byte, and the random values are 16-bit precision, where the last significant bit is used for the sign. Hence, a table of size 2^{16} bits is needed.

Additionally, one can perform the Fisher-Yates random shuffle (or Knuth shuffle) [15] in order to mask the relationship between the side channel and the secret information (the samples). Specifically, after sampling the matrices $\boldsymbol{S'}$ and $\boldsymbol{E''}$ their samples are shuffled. This countermeasure *might* be robust against our attack, given that each session key is specific to a single session establishment. However, in case the attacker succeeded to force the victim reusing the same session key, the countermeasure of random shuffling is not secure anymore and was proven to leak information [25]. An attacker needs a marginally larger, yet still practical number of samples to rearrange the coordinates and undo the shuffle.

To conclude, KEMs are important cryptographic routines for large-scale communication protocols. As ephemeral secrets are used in these protocols, the risk of being vulnerable to side-channel analysis can be underestimated. This is because the chance of an attack being successful decreases when the same key is used only once in a single execution of the scheme. In this paper, we validate that it is indeed crucial to examine the vulnerability of these schemes against single-trace attacks targeting the session key. As lattice-based key exchange protocols are already deployed in practical applications, their side-channel evaluation should play a role in the decision of their implementation choices. This paper examines the feasibility of side-channel analysis against KEMs using CDT Gaussian sampling and proves that the latter is still vulnerable to machine-learning side-channel attacks even in real-world circumstances. We demonstrate our results using FrodoKEM as an example, which is a NIST alternate candidate from the third round, and has been recommended by the German Federal

Office for Information Security(BSI) [9] and the Netherlands National Communications Security Agency (NBV) [1] for achieving quantum-safe communication. Our single-trace attack leads to the recovery of the complete session key.

Acknowledgment

The work described in this paper has been supported by the German Federal Ministry of Education and Research (BMBF) under the project Full Lifecycle Post-Quantum PKI - FLOQI (ID 16KIS1074) and under the project Aquorypt (ID 16KIS1022).

References

1. Netherlands National Communications Security Agency. Prepare for the threat of quantum-computers, 2022. `https://english.aivd.nl/publications/publications/2022/01/18/prepare-for-the-threat-of-quantumcomputers`.
2. Erdem Alkim, Joppe W. Bos, Léo Ducas, Patrick Longa, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Chris Peikert, and Ananth Raghunathan Douglas Stebil. Frodokem: Learning with errors key encapsulation. Github. `https://github.com/microsoft/PQCrypto-LWEKE`.
3. Furkan Aydin, Aydin Aysu, Mohit Tiwari, Andreas Gerstlauer, and Michael Orshansky. Horizontal side-channel vulnerabilities of post-quantum key exchange and encapsulation protocols. *ACM Trans. Embed. Comput. Syst.*, 20(6), oct 2021.
4. Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1006–1018, 2016.
5. Joppe W Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam. Fly, you fool! faster frodo for the arm cortex-m4. *Cryptology ePrint Archive*, 2018.
6. Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Provably weak instances of ring-lwe revisited, 05 2016.
7. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
8. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Bimodal lattice signature scheme (bliss). `https://wiki.strongswan.org/projects/strongswan/wiki/BLISS`.
9. Federal Office for Information Security (BSI). Bsi tr-02102-1: ”cryptographic mechanisms: Recommendations and key lengths” version: 2022-1, 2022. `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.html`.
10. NewAE Technology Inc. `https://www.mouser.com/datasheet/2/894/NAE-CW308-datasheet-1289269.pdf`.
11. Mehmet Sinan Inci, Berk Gulmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Cache attacks enable bulk key recovery on the cloud, 08 2016.

12. Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. `https://github.com/mupq/pqm4`.
13. Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking NIST PQC on ARM cortex-m4. *IACR Cryptol. ePrint Arch.*, page 844, 2019.
14. Suhri Kim and Seokhie Hong. Single trace analysis on constant time cdt sampler and its countermeasure. *Applied Sciences*, 8(10), 2018.
15. Donald E Knuth. *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.
16. Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 20–33, Cham, 2015. Springer International Publishing.
17. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag, Berlin, Heidelberg, 2007.
18. Soundes Marzougui and Juliane Krämer. Post-quantum cryptography in embedded systems, 2019.
19. Soundes Marzougui, Vincent Ulitzsch, Mehdi Tibouchi, and Jean-Pierre Seifert. Profiling side-channel attacks on dilithium: A small bit-fiddling leak breaks it all. Cryptology ePrint Archive, Paper 2022/106, 2022. `https://eprint.iacr.org/2022/106`.
20. Soundes Marzougui, Nils Wisiol, Patrick Gersch, Juliane Krämer, and Jean-Pierre Seifert. Machine-learning side-channel attacks on the galactics constant-time implementation of bliss, 2021.
21. Michele Mosca. Cybersecurity in an era with quantum computers: Will we be ready? *IEEE Security Privacy*, 16(5):38–41, 2018.
22. Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked ind-cca secure saber kem implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 676–707, 2021.
23. National Institute of standards and technology. Nist pqc standardization process. `https://csrc.nist.gov/Projects/post-quantum-cryptography`.
24. Sebastian Paul, Felix Schick, and Jan Seedorf. Tpm-based post-quantum cryptography: A case study on quantum-resistant and mutually authenticated tls for iot environments. In *The 16th International Conference on Availability, Reliability and Security*, ARES 2021, New York, NY, USA, 2021. Association for Computing Machinery.
25. Peter Pessl. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In *International Conference on Cryptology in India*, pages 153–170. Springer, 2016.
26. Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on cca-secure lattice-based pke and kems. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):307–335, Jun. 2020.
27. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), sep 2009.
28. Rhode and Schwarz. Kryptogeräte. `https://www.rohde-schwarz.com/de/produkte/aerospace-verteidigung-sicherheit/kryptogeraete_230846.html`.

29. Tobias Schneider, Clara Paglialonga, Tobias Oder, and Tim Güneysu. Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto. In *IACR International Workshop on Public Key Cryptography*, pages 534–564. Springer, 2019.
30. scikit learn. scikit-learn machine learning in python. `https://scikit-learn.org/stable/`.
31. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, oct 1997.
32. Bo-Yeon Sim, Jihoon Kwon, Joohee Lee, Il-Ju Kim, Tae-Ho Lee, Jaeseung Han, Hyojin Yoon, Jihoon Cho, and Dong-Guk Han. Single-trace attacks on message encoding in lattice-based kems. *IEEE Access*, 8:183175–183191, 2020.
33. Douglas Stebila and Michele Mosca. liboqs is an open source C library for quantum-safe cryptographic algorithms., Cortex-M4. `https://github.com/open-quantum-safe/liboqs`.
34. Vincent Quentin Ulitzsch, Shinjo Park, Soundes Marzougui, and Jean-Pierre Seifert. A post-quantum secure subscription concealed identifier for 6g. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '22, page 157–168, New York, NY, USA, 2022. Association for Computing Machinery.
35. Utimaco. What is a hardware security module (hsm). `https://utimaco.com/de/produkte/technologien/hardware-security-modules/what-hardware-security-module-hsm`.
36. H. Wang, M. Brisfors, S. Forsmark, and E. Dubrova. How diversity affects deep-learning side-channel attacks. In *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, pages 1–7, 2019.
37. Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 305–316, New York, NY, USA, 2012. Association for Computing Machinery.
38. Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. Facct: Fast, compact, and constant-time discrete gaussian sampler over integers. *IEEE Transactions on Computers*, 69(1):126–137, 2020.