Side-Channel Analysis of Saber KEM Using Amplitude-Modulated EM Emanations

1st Ruize Wang
Department of Electronic Systems
KTH Royal Institute of Technology
Stockholm, Sweden
ruize@kth.se

2nd Kalle Ngo
Department of Electronic Systems
KTH Royal Institute of Technology
Stockholm, Sweden
kngo@kth.se

3rd Elena Dubrova
Department of Electronic Systems
KTH Royal Institute of Technology
Stockholm, Sweden
dubrova@kth.se

***Abstract*—In the ongoing last round of NIST's post-quantum cryptography standardization competition, side-channel analysis of finalists is a main focus of attention. While their resistance to timing, power and near field electromagnetic (EM) side-channels has been thoroughly investigated, amplitude-modulated EM emanations has not been considered so far. The attacks based on amplitude-modulated EM emanations are more stealthy because they exploit side-channels intertwined into the signal transmitted by an on-chip antenna. Thus, they can be mounted on a distance from the device under attack. In this paper, we present the first results of an amplitude-modulated EM side-channel analysis of one of the NIST PQ finalists, Saber key encapsulation mechanism (KEM), implemented on the nRF52832 (ARM Cortex-M4) system-on-chip supporting Bluetooth 5. By capturing amplitude-modulated EM emanations during decapsulation, we can recover each bit of the session key with 0.91 probability on average.**

***Index Terms*—Public-key cryptography, Post-quantum cryptography, Saber KEM, LWE/LWR-based KEM, Side-channel attack, EM analysis, Deep learning**

I. Introduction

Today's public-key cryptographic schemes rely on the intractability of mathematical problems such as integer factorization or the discrete logarithm problem which can be efficiently solved using the Shor algorithm [1] on a large-scale quantum computer. Even if it may take many years for large-scale quantum computers to become a reality, the need for long-term security makes it necessary to investigate new solutions.

To address this issue, the National Institute of Standards and Technology (NIST) launched a competition for standardizing post-quantum cryptographic primitives (NIST PQ standardization project) starting in 2016 [2]. Candidate primitives are based on problems such as lattice problems and decoding problems for error correcting codes that are believed to be difficult for quantum computers. In rounds 1 and 2, the main focus of evaluation was on security and implementation aspects. Now the project is in the final round 3, where resistance to side-channel attacks is a main focus.

At present, lattice-based cryptography seems to be the most promising area in post-quantum cryptography, as three out of four finalists of NIST PQ for the primitive KEM are lattice-based. These three finalists are: an NTRU-based scheme NTRU [3], a Learning With Errors (LWE)-based scheme Kyber [4], and a Learning With Rounding (LWR)-based scheme Saber [5]. The hardness of these problems stems from the introduction of unknown noise into otherwise linear equations. In this paper, we focus on side-channel analysis on Saber KEM.

Previous work: Timing, power and near field EM side-channel attacks on software and hardware implementations of NIST PQ candidates have been presented in the past. In [6], a secret key recovery attack using a single power trace on three NIST post-quantum lattice-based finalists was proposed. In [7], a message (session key) recovery attack using a single power trace from an unprotected encapsulation part of several round 3 candidates, including Saber, was presented. In [8], near field EM message recovery attacks on some round 3 candidates, including Saber, were described. In [9], a secret key recovery attack on an unprotected Kyber using near field EM side-channels was demonstrated. In [10] similar ideas for timing attacks were considered.

In [11], near field EM secret key recovery attacks on unprotected implementations of three NIST PQ finalists, including Saber, were presented. It was also discussed how masked implementations can be broken by attacking each share individually. In [12], a deep learning-based side-channel attack on a first-order masked software implementation of Saber KEM which can recover the secret key from 24 power traces was demonstrated. In [13], it was shown that Saber's secret key can be recovered from 61,680 power traces even if masking is complemented with a shuffling countermeasure. In [14], power/near field EM secret key recovery attack on some round 3 candidates, including Saber, was described. This attack uses side-channel leakage during execution of the re-encryption step of decapsulation as a plaintext-checking oracle that tells whether the PKE decryption results is equivalent to the reference plaintext, or not.

While the resistance of NIST PQ finalists to timing, power and near field EM side-channels has been thoroughly investigated, amplitude-modulated EM emanations have not been considered so far. The attacks based on amplitude-modulated EM emanations are more stealthy because they exploit side-channels intertwined into the signal transmitted by an on-chip antenna [15]. Thus, they can be carried out on a distance from the device under attack. E.g. in [16], a successful attack on

AES-128 from 15 m distance was demonstrated.

Our contributions: In this paper, we present the first results of an amplitude-modulated EM side-channel analysis of the Saber KEM implemented in an ARM Cortex-M4 CPU in nRF52832 system-on-chip (SoC) supporting Bluetooth 5. By capturing amplitude-modulated EM emanations during the PKE decryption step of decapsulation, we can recover each bit of the session key with 0.91 probability on average. We analyze the specifics of amplitude-modulated EM emanations and discuss possibilities for improving the success probability.

The remainder of this paper is organized as follows. Section II gives background on the Saber design and EM emanations. In Section III presents details of the experimental setup. Sections IV and V describe how we train DL models and perform message recovery attack. Experimental results are shown in Section VI. Section VII concludes the paper.

II. Background

This section briefly describes Saber design and amplitude-modulated EM emanations. We explain the difference between the amplitude-modulated EM side-channels and power and near field EM side-channels.

A. Saber design

Saber is a package of cryptographic algorithms whose security relies on the hardness of the Module Learning With Rounding problem (Mod-LWR) [5]. It contains a CPA-secure public key encryption scheme, Saber.PKE, and a CCA-secure key encapsulation mechanism, Saber.KEM, based on a post-quantum version of the Fujisaki-Okamoto transform [17].

Pseudocodes of Saber.KEM and Saber.PKE are shown in Fig. 1 and 2, respectively. We follow the notation of [12].

The term $x \leftarrow \chi(S)$ is used to denote sampling x from a distribution χ over a set S. The uniform distribution is denoted by $\mathcal{U}$. The centered binomial distribution with parameter μ is denoted by β_μ, where μ is an even positive integer.

The functions $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{H}$ are SHA3-256, SHA3-512 and SHA3-256 hash functions, respectively. The gen is an extendable output function used to generate a pseudorandom matrix $\mathbf{A} \in R_q^{l \times l}$ from $seed_{\mathbf{A}}$. It is instantiated with SHAKE-128.

The bitwise right shift operation is denoted by "$\gg$". By performing the shift coefficient-wise, it is extended to polynomials and matrices. To enable for an efficient implementation, Saber design uses power of two moduli q, p, and T, namely $q = 2^{\epsilon_q}$, $p = 2^{\epsilon_p}$, and $T = 2^{\epsilon_T}$. Three constants are utilized to implement rounding operations using a simple bit shift: polynomials $h_1 \in R_q$ and $h_2 \in R_q$ with all coefficients being $2^{\epsilon_q - \epsilon_p - 1}$ and $2^{\epsilon_p - 2} - 2^{\epsilon_p - \epsilon_T - 1} + 2^{\epsilon_q - \epsilon_p - 1}$, respectively, and a constant vector $\mathbf{h} \in R_q^{l \times 1}$ in which each polynomial equals h_1.

Saber uses parameters $n = 256$, $l = 3$, $q = 2^{13}$, $p = 2^{10}$, $T = 2^4$, and $\mu = 8$. Its decryption failure probability is bounded by 2^{-136}.

B. Amplitude-modulated EM emanations

Near field EM side-channel attacks require a close proximity to the target device. Acquiring good quality near field EM emanations may require chip decapsulation [19]. Recently, a new type of side-channels in mixed-signal circuits, called *screaming channels* was discovered [15], which can be exploited at a distance from target device.

A mixed-signal circuit integrates a digital part and analog part. Cryptographic algorithms are executed in the digital part controlled by the internal system clock from the CPU, resulting in direct EM emanations. These emanations may propagate from the digital part to the analog part through substrate coupling [20], get mixed with the radio carrier inadvertently (amplitude-modulated), and then amplified and transmitted by the antenna [15]. This makes it possible to acquire EM emanations at a distance from the target device and recover the secret key from implementations of some cryptographic algorithms, e.g. AES-128 [15].

Such types of side-channels are difficult to mitigate because traditional countermeasures against near field EM leakage, such as shielding, do not cover an RF antenna. Hence, shielding cannot stop amplitude-modulated EM emanations. Thus, it is important to analyze NIST PQ finalists with respect to these side-channels.

III. Trace acquisition

This section describes how we captured EM emanations and located points of interest (PoI) in traces.

A. Experimental setup

At the transmitter side, the device under attack is an nRF52832 chip supporting Bluetooth 5 with a data transmission rate of 2Mbps. The nRF52832 contains a 32-bits ARM Cortex-M4 CPU running at 64MHz. It is mounted on an Nordic Semiconductors nRF52 DK development board. We use `nRF5_SDK_14.2.0_17b948a` for the radio setup.

The nRF52832 device is programmed to the C implementation of SABER from [18]. It does not contain any countermeasures against power/EM analysis. In our experiments, we use `gcc-arm-none-eabi-8-2018-q4-major` compiler with two different optimization levels: no optimization (-O0) and the highest level of optimization (-O3).

At the receiver side, we use an Ettus Research USRP N210 software defined radio (SDR) with the center receiving frequency set to $2f_{clock} + f_{\text{Bluetooth}} = 2.528$GHz, where $f_{\text{Bluetooth}} = 2.4$GHz is the Bluetooth channel center frequency and $f_{clock} = 64$MHz is the frequency of the CPU clock. We determined the center frequency of the receiver using the same method as in [15].

In our experiments, we use three different sampling frequencies: 5MHz, 12.5MHz and 25MHz. The latter is the maximum sampling frequency of USRP N210 SDR (limited by interface). The signal is transmitted to the receiver through an SMA coaxial cable.

The total equipment cost for the experiments is approximately 3,000 €.

```
Saber.PKE.KeyGen()
 1: $seed_{\mathbf{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$
 2: $\mathbf{A} = \mathsf{gen}(seed_{\mathbf{A}}) \in R_q^{l\times l}$
 3: $r \leftarrow \mathcal{U}(\{0,1\}^{256})$
 4: $\mathbf{s} \leftarrow \beta_\mu(R_q^{l\times 1}; r)$
 5: $\mathbf{b} = ((\mathbf{A}^T\mathbf{s} + \mathbf{h}) \mod q) \gg (\epsilon_q - \epsilon_p) \in R_p^{l\times 1}$
 6: return $(pk := (seed_{\mathbf{A}}, \mathbf{b}), sk := \mathbf{s})$

Saber.PKE.Dec($\mathbf{s}, (\boldsymbol{c_m}, \mathbf{b}')$)
 1: $v = \mathbf{b}'^T(\mathbf{s} \mod p) \in R_p$
 2: $m' = ((v + h_2 - 2^{\epsilon_p-\epsilon_T}\boldsymbol{c_m}) \mod p) \gg (\epsilon_p - 1) \in R_2$
 3: return $m'$
```

```
Saber.PKE.Enc($(seed_{\mathbf{A}}, \mathbf{b}), m; r$)
 1: $\mathbf{A} = \mathsf{gen}(seed_{\mathbf{A}}) \in R_q^{l\times l}$
 2: if r is not specified then
 3:     $r \leftarrow \mathcal{U}(\{0,1\}^{256})$
 4: end if
 5: $\mathbf{s}' \leftarrow \beta_\mu(R_q^{l\times 1}; r)$
 6: $\mathbf{b}' = ((\mathbf{A}\mathbf{s}' + \mathbf{h}) \mod q) \gg (\epsilon_q - \epsilon_p) \in R_p^{l\times 1}$
 7: $v' = \mathbf{b}^T(\mathbf{s}' \mod p) \in R_p$
 8: $\boldsymbol{c_m} = ((v' + h_1 - 2^{\epsilon_p-1}m) \mod p) \gg (\epsilon_p - \epsilon_T) \in R_T$
 9: return $(c := (\boldsymbol{c_m}, \mathbf{b}'))$
```

Fig. 1: Pseudocode of Saber.PKE from [5].

```
Saber.KEM.KeyGen()
 1: $(seed_{\mathbf{A}}, \mathbf{b}, \mathbf{s})$ = Saber.PKE.KeyGen()
 2: $pk = (seed_{\mathbf{A}}, \mathbf{b})$
 3: $pkh = \mathcal{F}(pk)$
 4: $z \leftarrow \mathcal{U}(\{0,1\}^{256})$
 5: return $(pk := (seed_{\mathbf{A}}, \mathbf{b}), sk := (z, pkh, pk, \mathbf{s}))$
```

```
Saber.KEM.Encaps($(seed_{\mathbf{A}}, \mathbf{b})$)
 1: $m \leftarrow \mathcal{U}(\{0,1\}^{256})$
 2: $(\hat{K}, r) = \mathcal{G}(\mathcal{F}(pk), m)$
 3: $c$ = Saber.PKE.Enc($pk, m; r$)
 4: $K = \mathcal{H}(\hat{K}, c)$
 5: return $(c, K)$
```

```
Saber.KEM.Decaps($(z, pkh, pk, \mathbf{s})$,c)
 1: $m'$ = Saber.PKE.Dec($\mathbf{s}, c$)
 2: $(\hat{K}', r') = \mathcal{G}(pkh, m')$
 3: $c'$ = Saber.PKE.Enc($pk, m'; r'$)
 4: if $c = c'$ then
 5:     return $K = \mathcal{H}(\hat{K}', c)$
 6: else
 7:     return $K = \mathcal{H}(z, c)$
 8: end if
```

Fig. 2: Pseudocode of Saber.KEM from [5].

```
void indcpa_kem_dec(char *sk, char *ct, char m[])
uint16_t v[N];
uint16_t sksv[K][N];
 1: BS2POLVECq(sk,sksv);
 2: SABER_un_pack(&ct, v);
 3: for (i = 0; i < N; ++i) do
 4:     v[i] = h2-(v[i]<<(EP-ET));
 5: end for
 6: VectorMul(ciphertext,sksv,v);
 7: for (i = 0; i < N; ++i) do
 8:     v[i] = (v[i]&(P-1))>>(EP-1);
 9: end for
    /* pack decrypted message m */
10: POL2MSG(v,m);

void POL2MSG(uint16_t *v, char *m)
 1: for (j = 0; j < BYTES; j++) do
 2:     m[j] = 0;
 3:     for (i = 0; i < 8; i++) do
 4:         m[j] = m[j]|(v[8*j+i]<<i);
 5:     end for
 6: end for
```

Fig. 3: C code of Saber.PKE.Dec() from [18].

B. Locating points of interest

The vulnerabilities of unprotected LWE/LWR-based PKE/KEMs have been investigated in previous work [7], [8], [11], [21]. One of them is *Incremental-Storage*, discovered in [11], which is caused by an incremental update of the decrypted message in memory during message decoding. The decoding operation (line 2 of Saber.PKE.Decrypt() at Fig. 1) computes the decrypted message bit by bit, mapping each polynomial coefficient into a corresponding message bit.

In Saber, two Incremental-Storage vulnerabilities are known [11]. The first is in the message decoding operation (line 8 of `indcpa_kem_dec()` at Fig. 3), where the message bits are computed and stored in the memory location `v[i]` in an unpacked fashion. The second is in `POL2MSG()` procedure (line 4 of `POL2MSG()` at Fig. 3), where the message bits are packed into a byte array in memory.

Fig. 4(a) shows an EM trace obtained by averaging 10K measurements made during the execution of Saber.KEM.Decaps() for random ciphertexts (complied with -O0 optimization level). In the beginning, we can see a segment representing the message decoding operation in which 256 message bits are decoded one-by-one. The segment following it is the `POL2MSG()` procedure in which message bits are packed into 32 bytes. By zooming in (Fig 4(b)), we can see the repeated patterns representing one byte processing by `POL2MSG()`. Note that different bytes look differently. This is because the number of data points corresponding to the processing of one byte by `POL2MSG()` differs for different message bytes. Similarly, in the message decoding operation, the number of points corresponding to the decoding of one bit differs for different message bits. A reason for this is that the sampling rate 25MHz is not a multiple of the CPU clock cycle 64 Mhz. Due to the differences among bits/bytes, we are not using the cut-and-join technique of [12] at the profiling stage.

We verified leakage points using the Test Vector Leakage Assessment (TVLA) method [22]. Fig. 6 shows the Welch's t-test results for the first eight message bits. We can see clear peaks in both, message decoding operation and `POL2MSG()` procedure, with `POL2MSG()` leakage being stronger. We can also see that `POL2MSG()` leakage is word-wise, showing that bytes are fetched from/loaded to the memory in groups of four. This is not surprising since the CPU of the target device is 32-bit.

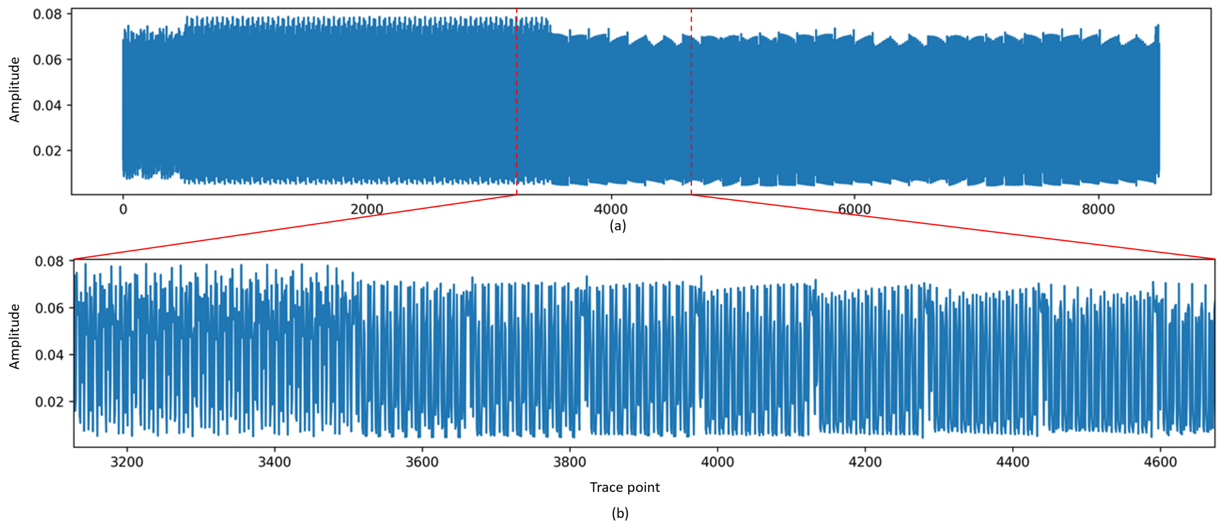

Fig. 4: An EM trace representing the execution of message decoding and `POL2MSG()` procedure.

```
TrainModels($N, in\_size$)
1: $\boldsymbol{m} = \{m_i \in \{0,1\}^{256} \mid m_i$ is random, $\forall i \in \{1,\ldots,N\}\}$
2: $\boldsymbol{c} = \{c_i \mid c_i =$ Saber.PKE.Enc$(pk, m_i; r_i), \forall i \in \{1,\ldots,N\}\}$
3: $\boldsymbol{\mathcal{T}}_{init} = \{\mathcal{T}_i \mid \mathcal{T}_i \Leftarrow$ Saber.KEM.Decaps$(c_i)$, $\forall i \in \{1,\ldots,N\}\}$
4: for each $j \in \{0,1,\ldots,255\}$ do
5:     $interval$ = SelectPoI$(j)$
6:     $\boldsymbol{\mathcal{T}} = \boldsymbol{\mathcal{T}}_{init}[:, interval]$
7:     $\boldsymbol{\mathcal{L}} = \{\ l(\mathcal{T}_i) \in \{0,1\} \mid l(\mathcal{T}_i) = m_i[j], \forall i \in \{1,\ldots,N\}\}$
8:     Train $\mathcal{N}_j : \mathbb{R}^{in\_size} \rightarrow \mathbb{I}$ on $(\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{L}})$
9: end for
10: return $\mathcal{N}_0, \mathcal{N}_1, \ldots, \mathcal{N}_{255}$
```

Fig. 5: Pseudocode of `TrainModels()` algorithm.

TABLE I: The MLP architecture; $in\_size = 720$ for -O0 and 75 for -O3.

Layer (Type)	Output Shape	Parameter #
Input	in_size	0
BatchNormalization1	in_size	2880
Dense1	128	92288
Dense2	128	16512
Dense3	32	4128
Dense4	16	528
Dropout1	16	0
Output	2	34

Total Parameters: 116,370

IV. Profiling stage

This section describes how we train neural networks at the profiling stage. Let $\boldsymbol{m} = \{m_1, m_2, \ldots, m_N\}$ be a set of messages $m_i \in \{0,1\}^{256}$ and $\boldsymbol{c} = \{c_1, c_2, \ldots, c_N\}$ be the set of corresponding ciphertexts $c_i =$ Saber.PKE.Enc$(pk, m_i; r_i)$. Let $\boldsymbol{\mathcal{T}} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N\}$ be a set of traces $\mathcal{T}_i \in \mathbb{R}^{|\mathcal{T}_i|}$ captured during the execution of Saber.KEM.Decaps() with c_i as input, $i \in \{1, 2, \ldots, N\}$.

A. Training strategy

The pseudocode of the profiling algorithm TrainModel() is shown in Fig. 5. It takes as input the number of training traces, N and the neural network's input size, *in_size*.

For each message bit $j \in \{0, 1, \ldots, 255\}$, we train an individual model $\mathcal{N}_j : \mathbb{R}^{|\mathcal{T}_i|} \rightarrow \mathbb{I}^2$, $\mathbb{I} := \{x \in \mathbb{R} \mid 0 \le x \le 1\}$, which takes as input a trace $\mathcal{T}_i \in \boldsymbol{\mathcal{T}}$ and produces as output a score vector $S_i = \mathcal{N}_j(\mathcal{T}_i)$ in which the value of the lth element, $s_{i,l}$, for $l \in \{0,1\}$, is the probability that $m_i[j] = l$ when the ciphertext c_i is applied as input. Message bits $m_i[j]$ are used as labels for training traces $\mathcal{T}_i \in \boldsymbol{\mathcal{T}}$ for $\mathcal{N}_j$.

The trace in Fig. 4(a) has 9500 data points, including 3013 points for message decoding operation and 4954 points for `POL2MSG()` procedure. As the input interval to $\mathcal{N}_j$, $interval$, we use a concatenation of two segments of the trace corresponding to the decoding of the message bit j and processing of the bit j by `POL2MSG()`. These segments are determined from the t-test. For the message decoding part, we use the segment containing the byte $\lfloor j/8 \rfloor$. For the `POL2MSG()` part, we use the segment containing the word $\lfloor j/32 \rfloor$. For -O0 optimization level, these segments contain 100 and 620 points, respectively.

B. Training details

The multilayer perceptron (MLP) architecture shown in Table I is used to train the neural networks in all experiments. We use categorical cross-entropy as a loss function. No trace normalization is applied for the input layer because all traces are captured through a coaxial cable, thus the amplitude deterioration during transmission can be neglected. *Nadam* optimizer with the learning rate 0.00005 and numerical stability constant *epsilon*=1e-8 is used. The batch size is set to 128, the number of epoch is 100 and the dropout rate is 0.2. We test all models and keep the best one only.

V. Attack stage

At the attack stage, majority voting is used to determine the final result. Majority voting is a known technique for

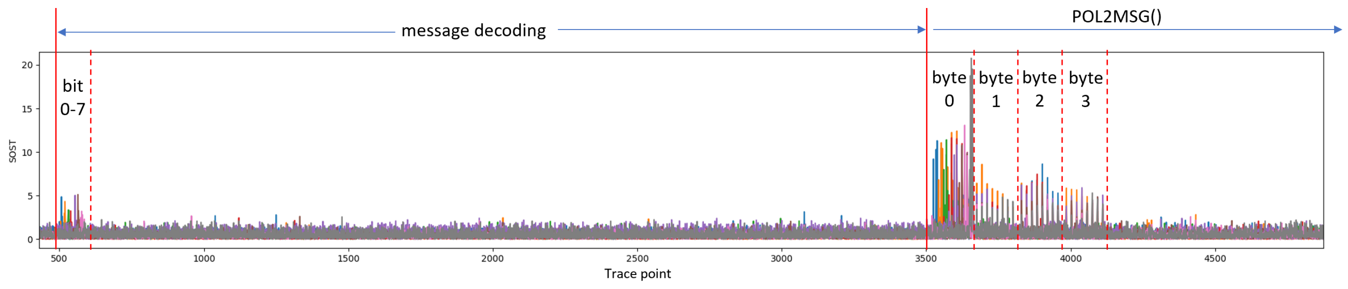

Fig. 6: T-test results for the message bits 0-7.

```
RecoverMessage($\mathcal{N}_0, \ldots, \mathcal{N}_{255}, c, M$)
1: $\boldsymbol{\mathcal{T}} = \{\mathcal{T}_i \mid \mathcal{T}_i \Leftarrow$ Saber.KEM.Decaps($c$), $\forall i \in \{1, \ldots, M\}\}$
2: for each $j \in \{0, 1, \ldots, 255\}$ do
3:     $interval$ = SelectPoI($j$)
4:     $m[j]$ = MajorityVoting($\mathcal{N}_j(\boldsymbol{\mathcal{T}}[interval])$)
5: end for
6: return $m = (m[0], \ldots, m[255])$
```

Fig. 7: Pseudocode of `RecoverMessage()` algorithm.

improving the success rate of side-channel attacks [12]. If p_j is the probability of recovering a message bit j from a single trace and errors are mutually independent, then the probability of recovering the bit j from M traces captured for the same input is given by:

$$p_{j,M} = \sum_{i=\lceil M/2 \rceil}^{M} \binom{M}{i} p_j^i (1-p_j)^{M-i},$$

where K is odd. For example, majority voting with the degree $M = 23$ can boost the probability from $p_j = 0.71$ to $p_{j,23} = 0.98$ if the errors are mutually independent. In reality, however, there is a dependency between errors in traces captured multiple times for the same input. This limits possibilities for improving the success rate by majority voting.

The pseudocode of the algorithm RecoverMessage() is shown in Fig. 7. In line 1, the set $\boldsymbol{\mathcal{T}} = \{\mathcal{T}_1, \ldots, \mathcal{T}_M\}$ is a set of traces captured for the same ciphertext c. To recover the message m encrypted in c, for each bit $j \in \{0, \ldots, 255\}$, we first recover each $m_i[j]$ from the score vector $S_i = \mathcal{N}_j(\mathcal{T}_i)$ as $m_i[j] = \arg\max(S_i)$, and then decide the final bit $m[j]$ by majority voting on M bits $m_i[j]$:

$$m[j] = \begin{cases} 1, & if \ \frac{1}{M} \sum_{i=1}^{M} m_i[j] > 0.5 \\ 0, & otherwise. \end{cases}$$

VI. Experimental results

This section presents our experimental results. To determine the best settings for the analysis, we first assess the effect of different sampling rates on side-channel leakage, as well as the need for pre-processing techniques such as trace averaging. Subsections VI-A and VI-B describe the findings, respectively. Subsection VI-C presents a message recovery attack based on the selected settings. We evaluate the impact of different compiler optimization levels, trace expansion and bit-level majority voting on the success rate. Finally, in subsection VI-D, we discuss why the our amplitude-modulated EM analysis of Saber KEM cannot achieve as high per-bit message recovery probability as power analysis of Saber KEM presented in [12].

A. The impact of sampling rate

The aim of this experiment is to investigate the side-channel leakage at different sampling rates. On one hand, the clock frequency of the ARM Cortex-M4 32-bit CPU in the nRF52832 device is 64MHz. On the other hand, the maximum SDR sampling rate achieved in our experiments is 25MHz. Thus, side-channel information about some clock cycles is lost. However, in previous work [15] the same type of SDR is used for amplitude-modulated EM analysis of a tiny AES-128 implementation on the same target device and sampling at 5MHz is sufficient to successfully recover the key. In this section, we analyse whether a 5MHz sampling rate is sufficient for Saber KEM and also consider 12.5MHz and 25MHz sampling rates.

Using the equipment described in Section III-A, we captured three 100K sets of traces $\boldsymbol{\mathcal{T}}_{5MHz}$, $\boldsymbol{\mathcal{T}}_{12.5MHz}$ and $\boldsymbol{\mathcal{T}}_{25MHz}$, containing the executions of both message decoding and `POL2MSG()` for random ciphertexts and a fixed secret key. There are 2000, 4500 and 8500 data points in each trace sampled at 5MHz, 12.5MHz and 25MHz respectively. The traces are captured from the implementation of Saber complied with the optimization level -O0.

We applied the t-test to the resulting sets of traces. For a given message bit $j \in \{0, 1, \ldots, 255\}$, each of the three sets was partitioned into two subsets, $\boldsymbol{\mathcal{T}}_0$ and $\boldsymbol{\mathcal{T}}_1$, containing traces $\mathcal{T}_i$ for which $m_i[j] = 0$ and $m_i[j] = 1$, respectively.

Table II lists the maximum sum of squared pairwise t-differences (SOST) values for the first eight message bits at different sampling rates. The average SOST at 25MHz is the largest, therefore, we use the sampling rate 25MHz in the rest of the experiments.

B. The effect of averaging

The aim of this experiment is to investigate how repeating the same measurement multiple times affects the signal-to-noise ratio (SNR). It is known that the random noise in measurements decreases as the square root of the number of averaged samples [23]. Thus, it is possible to enhance the SNR

TABLE II: Maximum SOST for the first 8 message bits.

bit	0	1	2	3	4	5	6	7	**avg**
5MHz	3.43	3.25	2.97	2.73	2.65	2.40	2.07	4.64	3.02
12.5MHz	2.71	3.08	2.65	2.21	1.73	2.90	3.00	9.91	3.52
25MHz	3.61	3.90	3.68	3.36	3.79	2.90	2.92	4.94	3.64

TABLE III: Maximum SNR ($\times 10^{-3}$) for the first 8 message bits.

bit	0	1	2	3	4	5	6	7	**avg**
$10K \times 1$	1.11	0.83	1.49	1.16	1.01	0.94	2.07	1.60	1.28
$10K \times 10$	1.30	1.61	2.16	1.14	1.34	0.88	1.69	10.78	2.61
$10K \times 100$	3.10	4.14	2.54	2.35	1.97	2.39	1.18	40.14	7.23

by averaging the measurements. We use the method of [24] to calculate the SNR as:

$$SNR = \frac{(E[\boldsymbol{\mathcal{T}}_0(t)] - \overline{E[\boldsymbol{\mathcal{T}}(t)]})^2 + (E[\boldsymbol{\mathcal{T}}_1(t)] - \overline{E[\boldsymbol{\mathcal{T}}(t)]})^2}{Var[\boldsymbol{\mathcal{T}}_0(t)] + Var[\boldsymbol{\mathcal{T}}_1(t)]},$$

where $E[\boldsymbol{\mathcal{T}}_l(t)]$ and $Var[\boldsymbol{\mathcal{T}}_l(t)]$ are the mean and variance of the set of traces $\boldsymbol{\mathcal{T}}_l$ with labels $l \in \{0, 1\}$ at the data point t.

We captured 10K traces with no averaging, $\boldsymbol{\mathcal{T}}_{10K\times1}$, with averaging 10, $\boldsymbol{\mathcal{T}}_{10K\times10}$, and averaging 100, $\boldsymbol{\mathcal{T}}_{10K\times100}$, for the same input ciphertext and a fixed key. The SNR results for the first eight message bits are shown in Table III. We can see that 10K traces with 100 repetitions give the best result. Therefore, we use $\times100$ averaged traces for training and testing of neural networks.

C. Message/session key recovery attack

In this section we evaluate the empirical probability of successful message recovery from EM emanations. In Saber KEM, a message recovery attack is easily turned into session key recovery. Given a recovered message m', one first computes $(\hat{K}', r') = \mathcal{G}(pkh, m')$ and then obtains the session key as $K = \mathcal{H}(\hat{K}', c)$ (see steps 2 and 5 of Saber.KEM.Decaps() in Fig. 2).

1) The impact of compiler optimization level: First, we evaluate the impact of different compiler optimization levels on the success rate. We compare two cases:

- Implementation without optimization (-O0).
- Implementation with the highest optimization level (-O3).

For each case, an MLP model with the architecture listed in Table I was trained using the method described in Section IV-A on a training set of size 14K containing the executions of both message decoding operation and `POL2MSG()` procedure. The traces were captured for random ciphertexts and a fixed secret key and pre-processed with $\times100$ averaging. The resulting models were tested by recovering the message from 20 traces (pre-processed with $\times100$ averaging) captured for the same ciphertext and applying the bit-wise majority voting. Table IV shows the results for the first eight bits.

For the traces captured without optimization, for bit 7 we can reach 100% accuracy. The average accuracy for the first eight bits is 85.1%. However, the average accuracy drops to 74.5% if the highest optimization level is used. In the latter case, none of the bits are higher than 90% accuracy.

TABLE IV: Empirical probability of recovering a message bit from 20 traces for different compiler optimization levels.

bit	0	1	2	3	4	5	6	7	**avg**
-O0	0.90	0.93	0.83	0.79	0.78	0.75	0.83	1.00	0.851
-O3	0.73	0.79	0.81	0.78	0.77	0.69	0.69	0.70	0.745

TABLE V: Empirical probability of recovering a message bit from 20 traces with and without training set expansion.

	bit	0	1	2	3	4	5	6	7	**avg**
Without expansion	14K	0.91	0.82	0.81	0.84	0.88	0.88	0.86	1.00	0.875
	23K	0.91	0.95	0.85	0.81	0.82	0.77	0.88	1.00	0.874
	32K	0.95	0.94	0.83	0.82	0.82	0.76	0.90	1.00	0.878
With expansion	14K	0.96	0.95	0.86	0.79	0.82	0.78	0.87	1.00	0.879
	23K	0.94	0.96	0.87	0.85	0.82	0.75	0.85	1.00	0.880
	32K	0.95	0.97	0.88	0.83	0.83	0.79	0.87	1.00	0.890

TABLE VI: Empirical probability of recovering a message bit from M traces.

M	1	10	20	30	40	50
14K	0.710	0.853	0.879	0.886	0.899	0.911
23K	0.700	0.840	0.890	0.898	0.915	0.921
32K	0.708	0.865	0.890	0.904	0.908	0.921
average	0.706	0.853	0.883	0.896	0.907	0.918

2) Training set expansion: We found it useful to expand the training set $\boldsymbol{\mathcal{T}}$ in the following way. For every pair of traces $(\mathcal{T}_i, \mathcal{T}_j)_{i \neq j} \in \boldsymbol{\mathcal{T}}$ with the same label l, we create a new trace $\mathcal{T}_k = \frac{\mathcal{T}_i + \mathcal{T}_j}{2}$ and add $\mathcal{T}_k$ to $\boldsymbol{\mathcal{T}}$ with the label l. In this way, we can expand sets of size 14K, 23K, 32K to 400K, 1M and 2M, respectively. This technique can be viewed as a type of data augmentation [25].

The MLP models were trained and tested as in the previous experiment. Table V shows the results for the implementation without optimization (-O0). The choice of implementation does not seem to matter for this experiment. We can see that trace expansion improves the empirical probability of recovering a message bit by 0.4%, 0.6% and 1.2%, respectively.

3) Bit-level majority voting: Table VI summarizes the empirical probability of recovering a message bit by the MLP models trained with the trace sets of size 14K, 23K and 32K and tested with a different majority voting degree M. We can see that $M = 50$ gives the highest accuracy, 0.921. After $M = 50$, the curve flattens and there is no significant improvement. In the Appendix we show the empirical probabilities of recovering all 256 message bits by the MLP models trained with the expanded 32K trace set and tested with $M = 50$ majority voting degree. The average accuracy is 0.91.

D. PoIs analysis

The SDR used in our experiments is capable of a maximum sampling rate of 50Mhz, but due to interface limitations, we

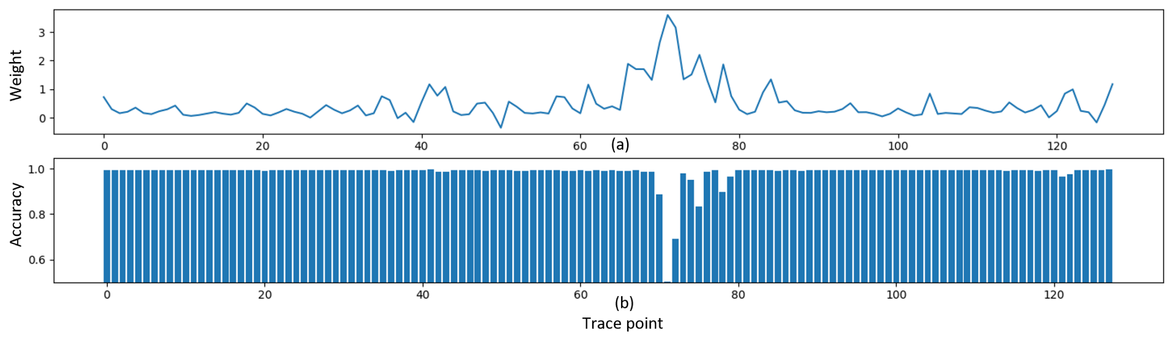

Fig. 8: (a) The weights of the input layer of the model trained on `poly_A2A()`; (b) The average test accuracy with single-bit erasures.

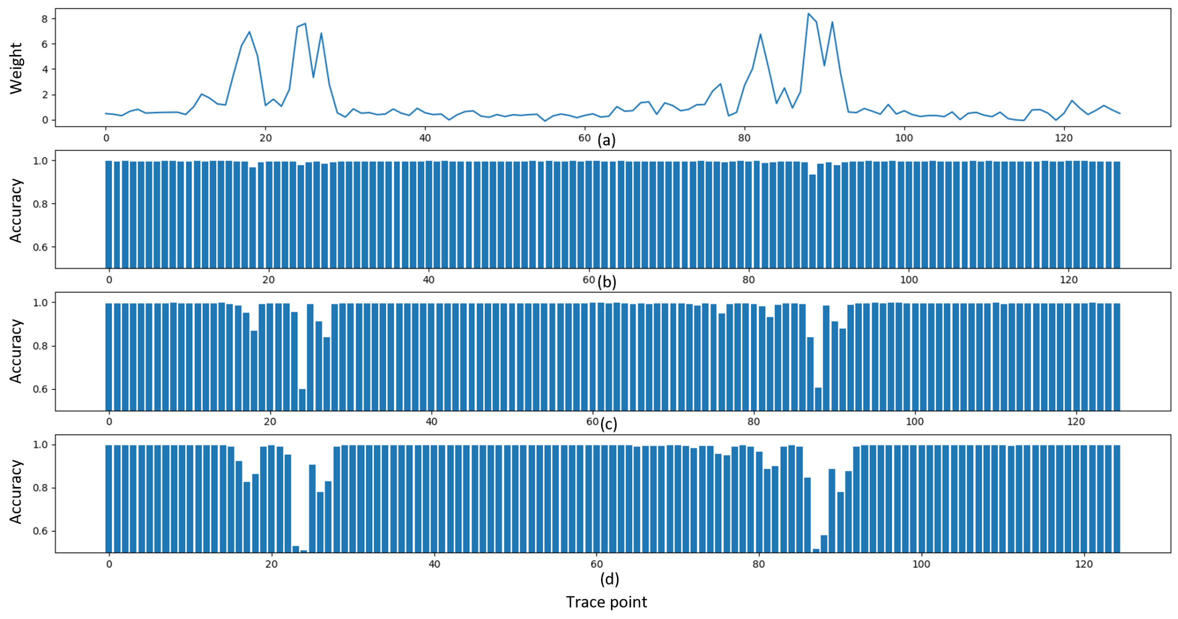

Fig. 9: (a) The weights of the input layer of the model trained on `POL2MSG()`; The average test accuracy: (b) with single-bit erasures; (c) with double-bit erasures; (d) with triple-bit erasures.

were limited to 25Mhz. This means that we sample 0.39 points per clock cycle from a CPU running at 64MHz. In this section we analyse whether this is a reason why we cannot reach a higher message recovery probability. For a comparison, in the attack on Saber in [12], the empirical probability of recovering a message bit from a single power trace is 0.997.

We used the publicly available trace data set and neural network models from [12] captured from a masked Saber KEM implementation run on the same target processor, ARM Cortex-M4, but another device, CW308T-STM32F4. In the experiments in [12], one point per clock cycle is captured. Thus, the traces contain side-channel information from each clock cycle. The trace data set from [12] includes segments representing the execution of `POL2MSG()` and `poly_A2A()` procedures. The implementation of `POL2MSG()` is the same as in the unprotected implementation which we use. The procedure `poly_A2A()` is a masked counterpart of the message decoding operation.

To evaluate the importance on PoIs in these procedures, we tested the models $\mathcal{N}^{P2M}$ and $\mathcal{N}^{A2A}$ trained on `POL2MSG()` and `poly_A2A()` traces segments, respectively, as follows. Let $\boldsymbol{\mathcal{T}}$ be a test set in which each trace $\mathcal{T}_i$ has n data points, $\mathcal{T}_i = (\tau_1, \ldots, \tau_n)$, where $\tau_t \in \mathbb{R}$ for all $t \in \{1, \ldots, n\}$. Let $\mathcal{T}_i|_{\tau_t=0}$ denote $\mathcal{T}_i$ with the data point τ_t being set to 0. In other words we "erase" from $\mathcal{T}_i$ the side-channel information captured at the time point (clock cycle) t. We do this to evaluate the contribution of each data point to the attack's success probability.

Fig 8 shows the results for the model $\mathcal{N}^{A2A}$ tested on the message bit 1. The top picture illustrates the distribution of weights of the input layer of $\mathcal{N}^{A2A}$. The weights determine relative contribution of each input to the model's decision. The bottom picture shows the average test accuracy for $\mathcal{N}^{A2A}(\mathcal{T}_i|_{\tau_t=0})$ for all $t \in \{1, \ldots, n\}$. Without any erasures, the test accuracy of $\mathcal{N}^{A2A}(\mathcal{T}_i)$ is 0.992. We can see that there is one data point, 71 (with the largest weight), whose erasure drops the the test accuracy to 0.531, close to a random guess. The erasure of its neighbours, 70 and 72, drop the test accuracy to 0.881 and 0.747, respectively.

Fig 9 shows similar results for the model $\mathcal{N}^{P2M}$ tested on the message bit 1, except that here we also show erasures affecting two and three adjacent data points in Fig 9(c) and (d) respectively. Without any erasures, the average test accuracy of $\mathcal{N}^{P2M}(\mathcal{T}_i)$ is 0.995. The single-bit erasures reduce the test accuracy at most by 7%, at point 88. Double-bit erasures reduce the test accuracy to 0.601 at the point 24. Only triple-bit erasures drop the test accuracy to 0.512 at the point 24.

From Fig. 8 we can conclude that, in order to fully exploit leakage from the message decoding operation of Saber, it is necessary to capture at least one data points per clock cycle. The interface limit of the SDR in our current experiments does not allow for this. Note that an SDR with the sampling frequency higher than 64MHz, such as USRP-2944R, costs about 9,000€ i.e. it is beyond the reach of a low-budget attacker.

From Fig. 9 it also becomes evident why in Fig 6 we get higher t-test peaks for `POL2MSG()` than for the message decoding (in [12] it is the opposite). Since we capture 0.39 points per clock cycle, we obtain at least one of the three essential leakage points of `POL2MSG()`.

VII. Conclusion

We demonstrated a side-channel attack on a software implementation of the Saber KEM based on amplitude-modulated EM emanations and discussed several ways for improving the success probability. Future work includes finding methods to mitigate amplitude-modulated EM side channels.

VIII. Acknowledgements

This work was supported in part by the Swedish Civil Contingencies Agency (grant No. 2020-11632) and the Swedish Research Council (grant No. 2018-04482).

References

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[2] NIST, "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process," 2016, https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[3] C. Chen *et al.*, "NTRU algorithm specifications and supporting documentation," 2020.

[4] P. Schwabe *et al.*, "CRYSTALS-Kyber algorithm specifications and supporting documentation," *https://csrc.nist.gov/projects/postquantum-cryptography/round-3-submissions*, 2020.

[5] J. D'Anvers *et al.*, "Saber algorithm specifications and supporting documentation," 2020.

[6] E. Karabulut, E. Alkim, and A. Aysu, "Single-trace side-channel attacks on ω-small polynomial sampling: With applications to NTRU, NTRU Prime, and CRYSTALS-Dilithium," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2021, pp. 35–45.

[7] B.-Y. Sim, J. Kwon, J. Lee, I.-J. Kim, T.-H. Lee, J. Han, H. Yoon, J. Cho, and D.-G. Han, "Single-trace attacks on message encoding in lattice-based KEMs," *IEEE Access*, vol. 8, pp. 183 175–183 191, 2020.

[8] P. Ravi, S. Sinha Roy, A. Chattopadhyay, and S. Bhasin, "Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs," vol. 2020, pp. 307–335, Jun. 2020.

[9] Z. Xu, O. M. Pemberton, S. S. Roy, D. Oswald, W. Yao, and Z. Zheng, "Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of Kyber," *IEEE Transactions on Computers*, 2021.

[10] Q. Guo, T. Johansson, and A. Nilsson, "A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM," in *Annual International Cryptology Conference*. Springer, 2020, pp. 359–386.

[11] P. Ravi, S. Bhasin, S. S. Roy, and A. Chattopadhyay, "On exploiting message leakage in (few) NIST PQC candidates for practical message recovery attacks," *IEEE Transactions on Information Forensics and Security*, 2021.

[12] K. Ngo, E. Dubrova, Q. Guo, and T. Johansson, "A side-channel attack on a masked IND-CCA secure Saber KEM implementation," *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pp. 676–707, 2021.

[13] K. Ngo, E. Dubrova, and T. Johansson, "Breaking masked and shuffled CCA secure Saber KEM by power analysis," in *Proc. of the 5th Workshop on Attacks and Solutions in Hardware Security*, 2021, pp. 51–61.

[14] R. Ueno, K. Xagawa, Y. Tanaka, A. Ito, J. Takahashi, and N. Homma, "Curse of re-encryption: A generic power/em analysis on post-quantum KEMs," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 296–322, 2022.

[15] G. Camurati, S. Poeplau, M. Muench, T. Hayes, and A. Francillon, "Screaming channels: When electromagnetic side channels meet radio transceivers," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 163–177.

[16] G. Camurati, A. Francillon, and F.-X. Standaert, "Understanding screaming channels: From a detailed analysis to improved attacks," *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pp. 358–401, 2020.

[17] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Annual international cryptology conference*. Springer, 1999, pp. 537–554.

[18] M. V. Beirendonck, J.-P. D'anvers, A. Karmakar, J. Balasch, and I. Verbauwhede, "A side-channel-resistant implementation of Saber," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 17, no. 2, pp. 1–26, 2021.

[19] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2001, pp. 251–261.

[20] S. Bronckers, G. Van der Plas, and Y. Rolain, *Substrate noise coupling in analog/RF circuits*. Artech House, 2010.

[21] D. Amiet, A. Curiger, L. Leuenberger, and P. Zbinden, "Defeating NewHope with a single trace," in *International Conference on Post-Quantum Cryptography*. Springer, 2020, pp. 189–205.

[22] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," in *NIST Mon-Invasive Attack Testing Workshop*, 2011.

[23] A. V. Oppenheim, J. R. Buck, and R. W. Schafer, *Discrete-time signal processing. Vol. 2*. Upper Saddle River, NJ: Prentice Hall, 2001.

[24] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas, "Deep learning for side-channel analysis and introduction to ASCAD database," *J. of Cryptographic Engineering*, vol. 10, no. 2, pp. 163–188, 2020.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

APPENDIX

TABLE VII: Empirical probability to recover a message bit from 50 traces (average for 100 messages).

	bit								
byte	0	1	2	3	4	5	6	7	**avg**
0	0.98	0.98	0.88	0.88	0.87	0.84	0.93	1.00	0.92
1	0.94	0.91	0.89	0.87	0.88	0.89	0.91	1.00	0.91
2	0.93	0.90	0.94	0.83	0.91	0.85	0.89	1.00	0.91
3	0.92	0.90	0.91	0.85	0.90	0.87	0.91	1.00	0.91
4	0.98	0.99	1.00	0.88	0.92	0.84	0.88	1.00	0.94
5	0.94	0.95	0.89	0.91	0.89	0.82	0.89	1.00	0.91
6	0.91	0.94	0.91	0.90	0.92	0.90	0.90	1.00	0.92
7	0.88	0.97	0.84	0.86	0.89	0.82	0.92	1.00	0.90
8	0.98	0.91	0.92	0.97	0.90	0.90	0.85	1.00	0.93
9	0.95	0.94	0.95	0.84	0.93	0.82	0.86	1.00	0.91
10	0.89	0.89	0.82	0.86	0.89	0.84	0.96	1.00	0.89
11	0.90	0.91	0.88	0.81	0.86	0.88	0.93	1.00	0.90
12	0.98	0.96	0.95	0.83	0.87	0.85	0.87	1.00	0.91
13	0.91	0.88	0.86	0.86	0.94	0.86	0.88	1.00	0.90
14	0.91	0.93	0.93	0.84	0.93	0.83	0.93	1.00	0.91
15	0.90	0.89	0.88	0.83	0.89	0.85	0.91	1.00	0.89
16	0.97	1.00	0.95	0.87	0.97	0.87	0.89	1.00	0.94
17	0.91	0.91	0.96	0.86	0.90	0.91	0.90	1.00	0.92
18	0.88	0.91	0.92	0.82	0.89	0.88	0.91	1.00	0.90
19	0.86	0.88	0.86	0.81	0.75	0.80	0.71	1.00	0.83
20	0.97	0.99	0.97	0.90	0.96	0.89	0.91	1.00	0.95
21	0.92	0.89	0.91	0.86	0.87	0.84	0.91	1.00	0.90
22	0.86	0.91	0.92	0.84	0.86	0.86	0.97	1.00	0.90
23	0.92	0.92	0.91	0.85	0.85	0.87	0.94	1.00	0.91
24	0.94	0.97	0.92	0.87	0.91	0.86	0.93	1.00	0.93
25	0.94	0.92	0.94	0.77	0.91	0.89	0.83	1.00	0.90
26	0.91	0.90	0.89	0.82	0.88	0.86	0.94	1.00	0.90
27	0.93	0.95	0.92	0.85	0.87	0.88	0.93	1.00	0.92
28	0.97	0.99	0.96	0.90	0.89	0.88	0.92	1.00	0.94
29	0.91	0.87	0.89	0.89	0.90	0.85	0.89	1.00	0.90
30	0.91	0.90	0.85	0.90	0.92	0.80	0.89	1.00	0.90
31	0.92	0.93	0.86	0.88	0.88	0.85	0.94	1.00	0.91
avg	0.93	0.93	0.91	0.86	0.89	0.86	0.90	1.00	**0.91**