

Making Biased DL Models Work: Message and Key Recovery Attacks on Saber Using Amplitude-Modulated EM Emanations

Ruize Wang, Kalle Ngo and Elena Dubrova

KTH Royal Institute of Technology, Stockholm, Sweden
 {ruize,kngo,dubrova}@kth.se

Abstract. Creating a good deep learning (DL) model is an art which requires expertise in DL and a large set of labeled data for training neural networks. Neither is readily available. In this paper, we introduce a method which enables us to achieve good results with bad DL models. We use simple multilayer perceptron (MLP) networks, trained on a small dataset, which make strongly biased predictions if used without the proposed method. The core idea is to extend the attack dataset so that at least one of its traces has the ground truth label to which the models are biased towards. The effectiveness of the presented method is demonstrated by attacking an ARM Cortex-M4 CPU implementation of Saber KEM, a finalist of the NIST post-quantum cryptography standardization project, on a nRF52832 system-on-chip supporting Bluetooth 5, using amplitude-modulated EM emanations. Previous amplitude-modulated EM emanation-based attacks on Saber KEM could not recover its messages with a sufficiently high probability. We recover messages with the probability 1 from the profiling device and with the probability 0.74 from a different device. Using messages recovered from chosen ciphertexts, we extract the secret key of Saber KEM.

Keywords: Public-key cryptography · Post-quantum cryptography · Saber KEM · LWE/LWR-based KEM · Side-channel attack · EM analysis

1 Introduction

Amplitude-modulated electromagnetic (EM) emanations are a type of side-channels which occur in mixed-signal chips with an on-board antenna. As a result of various coupling effects, signals from computations in the digital part of the chip may be modulated by the CPU clock signal, leak to the analog part of the chip, modulated again by the radio-frequency block, and eventually transmitted by the antenna.

Side-channel attacks based on amplitude-modulated EM emanations are more stealthy than power or near-field EM attacks because the signal transmitted by the on-chip antenna escapes hardware-level countermeasures like decoupling capacitors (used to smooth sharp changes in power supply voltage) and Faraday shields (used to block EM fields). Furthermore, since amplitude-modulated EM emanations are intertwined into the carrier signal, they can be captured at a considerably farther distance than the near-field EM side-channels. For example, in [CFS20], a successful attack on AES on 15 m distance from the device under attack was demonstrated.

However, amplitude-modulated EM emanations are much weaker than power and near-field EM side-channels. They require expensive equipment to capture and typically need post-processing by averaging multiple repeated measurements to increase the signal-

to-noise ratio. For example, 500 and 1000 measurements representing the same encryption were averaged in the attacks on AES presented in [CPM⁺18] and [CFS20], respectively.

Such excessive repetitions are undesirable in profiling deep learning (DL)-based side-channel attacks because they increase the size of training and attack sets by the corresponding factor. While the attack set is typically small, the training set is large. Minimizing the size of the latter is particularly important in the attacks on public key encryption algorithms since, in this case, the device under attack can be used for profiling [NDGJ21] (since the public key is known). Profiling on the device under attack eliminates the problem of device intra-variability and maximizes the prediction accuracy of DL models. If the secret to be recovered is large, achieving high prediction accuracy is crucial. For example, the secret messages of Saber KEM, a finalist of NIST post-quantum cryptography (PQC) project [NIS16], which is the focus of this paper, is 256-bit. So, if the message recovery is performed bit-by-bit, the model prediction accuracy should be at least 0.997 in order to get $0.997^{256} = 0.51$ message recovery probability. It is very difficult to achieve 0.997 prediction accuracy unless the model is trained on the device under attack, or a device manufactured in the same batch [NDGJ21].

If access time to the device under attack is an issue, the attacker who wants to use the device under attack for profiling faces the problem of training DL models on a small dataset. This is not an easy task, especially if the the number of classes to be distinguished is large, e.g. 256 classes in a byte-level classification with one-hot encoding. In such cases, training on a small dataset usually results in biased models which predict different classes non-uniformly. It has been observed that some labels might be strongly preferred [BF19].

Our contributions: In this paper, we introduce a method which makes it possible to recover messages of Saber KEM with a high probability using biased DL models trained on a small dataset. To recover the message m encrypted into a ciphertext c , we extend the attack set from a single trace, captured with c as input, to 256 traces, captured with c_e as inputs, for all $e \in \{0, 1, \dots, 255\}$. The modified ciphertexts c_e are constructed so that they decrypt to messages m_e in which the error e is injected into each byte of m . As a result, at least one of the traces in the attack set has the ground truth label to which the DL models are biased. The errors are injected using the bit-flipping technique from [RBRC21].

Our experimental results on a software implementation of Saber KEM in an ARM Cortex-M4 CPU in the nRF52832 system-on-chip supporting Bluetooth 5 show that the presented multiple-bit error injection method allows us to recover complete messages with the probability 1 from the profiling device and with the probability 0.74 from a different device. Using messages recovered for chosen ciphertexts constructed by method [NDGJ21], we successfully extract the secret key of Saber KEM. This is a significant improvement over the first amplitude-modulated EM emanation-based attack on Saber KEM [WND22]. The attack [WND22] recovers each message bit with the average probability of 0.91. Hence, it cannot recover the complete message successfully in the majority of cases.

Paper organization: The rest of this paper is organized as follows. Section 2 describes previous work related to the side-channel analysis of Saber KEM. Section 3 gives background on Saber design and its known vulnerabilities. Section 4 presents the experimental setup. Sections 5 and 6 describe how we train neural networks and perform message and secret key recovery attacks. Experimental results are summarized in Section 7. Section 8 concludes the paper.

2 Previous work

Since the launch of NIST PQC standardization project in 2016 [NIS16], timing, power and near field EM side-channel attacks on software and hardware implementations of NIST PQC candidates have received considerable attention. Three out of four finalists of NIST PQC are based on lattice problems: an NTRU-based scheme NTRU [C⁺20], a

<pre> Saber.PKE.KeyGen() 1: $seed_A \leftarrow \mathcal{U}(\{0, 1\}^{256})$ 2: $\mathbf{A} = \text{gen}(seed_A) \in R_q^{l \times l}$ 3: $r \leftarrow \mathcal{U}(\{0, 1\}^{256})$ 4: $\mathbf{s} \leftarrow \beta_\mu(R_q^{l \times 1}; r)$ 5: $\mathbf{b} = ((\mathbf{A}^T \mathbf{s} + \mathbf{h}) \bmod q) \gg (\epsilon_q - \epsilon_p) \in R_p^{l \times 1}$ 6: return $(pk := (seed_A, \mathbf{b}), sk := \mathbf{s})$ Saber.PKE.Dec($\mathbf{s}, (\mathbf{c}_m, \mathbf{b}')$) 1: $v = \mathbf{b}'^T (\mathbf{s} \bmod p) \in R_p$ 2: $m' = ((v + h_2 - 2^{\epsilon_p - \epsilon_T} \mathbf{c}_m) \bmod p) \gg (\epsilon_p - 1) \in R_2$ 3: return m' </pre>	<pre> Saber.PKE.Enc($(seed_A, \mathbf{b}), m; r$) 1: $\mathbf{A} = \text{gen}(seed_A) \in R_q^{l \times l}$ 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: $\mathbf{b}' = ((\mathbf{A} \mathbf{s}' + \mathbf{h}) \bmod q) \gg (\epsilon_q - \epsilon_p) \in R_p^{l \times 1}$ 7: $v' = \mathbf{b}'^T (\mathbf{s}' \bmod p) \in R_p$ 8: $\mathbf{c}_m = ((v' + h_1 - 2^{\epsilon_p - 1} m) \bmod p) \gg (\epsilon_p - \epsilon_T) \in R_T$ 9: return $(c := (\mathbf{c}_m, \mathbf{b}'))$ </pre>
--	---

Figure 1: Pseudocode of Saber.PKE [D⁺20].

<pre> Saber.KEM.KeyGen() 1: $(seed_A, \mathbf{b}, \mathbf{s}) = \text{Saber.PKE.KeyGen}()$ 2: $pk = (seed_A, \mathbf{b})$ 3: $pkh = \mathcal{F}(pk)$ 4: $z \leftarrow \mathcal{U}(\{0, 1\}^{256})$ 5: return $(pk := (seed_A, \mathbf{b}), sk := (z, pkh, pk, \mathbf{s}))$ Saber.KEM.Encaps($(seed_A, \mathbf{b})$) 1: $m \leftarrow \mathcal{U}(\{0, 1\}^{256})$ 2: $(\hat{K}, r) = \mathcal{G}(\mathcal{F}(pk), m)$ 3: $c = \text{Saber.PKE.Enc}(pk, m; r)$ 4: $K = \mathcal{H}(\hat{K}, c)$ 5: return (c, K) </pre>	<pre> Saber.KEM.Decaps($(z, pkh, pk, \mathbf{s}), c$) 1: $m' = \text{Saber.PKE.Dec}(\mathbf{s}, c)$ 2: $(\hat{K}', r') = \mathcal{G}(pkh, m')$ 3: $c' = \text{Saber.PKE.Enc}(pk, m'; r')$ 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 </pre>
---	--

Figure 2: Pseudocode of Saber.KEM [D⁺20].

Learning With Errors (LWE)-based scheme Kyber [S⁺20], and a Learning With Rounding (LWR)-based scheme Saber [D⁺20]. The fourth finalist, Classic McEliece [BCL⁺17], is based on decoding problems for error correcting codes (ECC). These problems are believed to be difficult for large-scale quantum computers.

In [SKL⁺20], a message recovery attack using a single power trace from an unprotected encapsulation part of several round 3 candidates, including Saber, was presented. In [RSRCB20], near field EM message recovery attacks on some round 3 candidates, including Saber, were described. In [GJN20], timing attacks were considered.

In [RBRC21], near field EM secret key recovery attacks on unprotected implementations of three NIST PQ finalists, including Saber, were presented. It was shown how masked implementations can be broken by attacking each share individually. In [NDGJ21], message and secret key recovery attacks on a first-order masked implementation of Saber KEM through DL-based power analysis were demonstrated. In [NDJ21], it was shown that it is possible to recover Saber secret key from 61,680 power traces even if masking is complemented with a shuffling countermeasure. In [UXT⁺22], power/near field EM secret key recovery attacks 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 are equivalent to the reference plaintext, or not.

The resistance of NIST PQC finalists to amplitude-modulated EM emanations has been investigated much less compared to timing, power and near-field EM side-channels. The first attack on Saber KEM has been recently presented in [WND22]. This attack

uses the same C implementation of Saber KEM and the same target device as in our experiments. The C implementation is compiled with the optimization level `-O0`. Using amplitude-modulated EM emanations during the PKE decryption step of decapsulation, each bit of a message is recovered with probability of 0.91 on average if the profiling and the attack devices are the same. So, the probability to recover a complete message is very small, $0.91^{256} = 0.33 \cdot 10^{-12}$. In contrast, for the optimization level `-O0`, the presented multiple-bit error injection method can recover a complete message with the probability 1 from the profiling device and with the probability 0.74 from a different device. We also show successful results for `-O3` optimization level.

The presented multiple-bit error injection method makes use of the bit-flip technique introduced in [RBRC21] for breaking implementations of LWE/LWR-based PKE/KEMs protected by the shuffling countermeasure. In [RBRC21], single message bits are flipped in order to quantify the effect of the change on the message Hamming weight (HW). The decrease/increase of the HW implies that the original message bit has the value 1/0. Note that the purpose of flipping bits in [RBRC21] is quite different from the one in the presented method. We inject multiple-bit errors to match the modified message bytes with labels preferred by the DL models.

3 Background

This section describes Saber design and vulnerabilities discovered in its software implementations so far.

3.1 Saber design

Saber consists of a CPA-secure public key encryption scheme, Saber.PKE, and a CCA-secure key encapsulation mechanism, Saber.KEM, which is based on a post-quantum version of the Fujisaki-Okamoto transform [FO99]. The security of Saber algorithms relies on the Module Learning With Rounding (Mod-LWR) problem’s difficulty [D⁺20].

Fig. 1 and 2 show pseudocodes of Saber.PKE and Saber.KEM algorithms, respectively. We use the same notation as in [NDGJ21]. Saber.PKE contains three algorithms: key generation, Saber.PKE.KeyGen; encryption, Saber.PKE.Enc; and decryption, Saber.PKE.Dec, as shown in Fig. 1. Saber.KEM also contains three algorithms: key generation, Saber.KEM.KeyGen; encapsulation, Saber.KEM.Encaps; and decapsulation, Saber.KEM.Decaps, as shown in Fig. 2.

Let \mathbb{Z}_q be the ring of integers modulo a positive integer q and R_q be the quotient ring $\mathbb{Z}_q[X]/(X^n + 1)$. The rank of the module and the rounding modulus are denoted by p and l , respectively.

The term $x \leftarrow \chi(S)$ denotes 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 samples of β_μ are in the range $[-\mu/2, \mu/2]$. Its probability mass function is given by $P[x] = \frac{\mu!}{(\mu/2+x)!(\mu/2-x)!} 2^{-\mu}$. The term $\beta_\mu(R_q^{l \times k}; r)$ induces a matrix in $R_q^{l \times k}$ in which the coefficients of polynomials of R_q are sampled deterministically from β_μ using seed r .

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 `seedA`. 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 used to implement rounding operations through a bit shift: polynomials

```

void indcpa_kem_dec(char *sk, char *ct, char m[]) void POL2MSG(uint16_t *v, char *m)
uint16_t v[N];                                1: for (j = 0; j < BYTES; j++) do
uint16_t sksv[K][N];                          2:   m[j] = 0;
1: BS2POLVECq(sk,sksv);                       3:   for (i = 0; i < 8; i++) do
2: SABER_un_pack(&ct, v);                     4:     m[j] = m[j] | (v[8*j+i]<<i);
3: for (i = 0; i < N; ++i) do                 5:   end for
4:   v[i] = h2-(v[i]<<(EP-ET));                6: end for
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);

```

Figure 3: C code of Saber.PKE.Dec [BDK⁺21].

$h_1 \in R_q$ 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 is equal to h_1 .

In the final round of NIST PQC project, three sets of parameters are proposed in Saber documentation [D⁺20] for the security levels of NIST-I, NIST-III, and NIST-V: LightSaber, Saber and FireSaber, respectively. In this paper we focus on Saber which has 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} .

3.2 Vulnerabilities in Saber

Several vulnerabilities have been discovered in software implementations of Saber, including *Incremental-Storage* vulnerability [RBRC21], weakness of re-encryption operation in Fujisaki-Okamoto transform [UXT⁺22] and weakness of polynomial multiplication [MBB⁺22].

In [RBRC21], two different types of Incremental-Storage vulnerabilities in two procedures of Saber were reported:

- *Bitwise-Storage* in message decoding operation (line 2 of Saber.PKE.Decrypt() in Fig. 1), in which the message bits are computed and stored in the memory location $v[i]$ in an unpacked fashion (line 8 of `indcpa_kem_dec()` in Fig. 3).
- *Byte-wise-Storage* in POL2MSG() procedure in which every eight message bits are packed into a byte and stored in memory (line 4 of POL2MSG() in Fig. 3).

In this paper, we exploit the Byte-wise-Storage vulnerability of POL2MSG() procedure and demonstrate that it allows us to recover messages and the secret key of Saber from the amplitude-modulated EM emanations.

4 Trace acquisition

This section describes how we captured amplitude-modulated EM emanations, pre-processed resulting traces, and selected intervals of interest.

4.1 Experimental setup

The experimental setup is shown in Fig. 4. We use the same target device and the same equipment for traces acquisition as in the side-channel attack presented in [WND22].

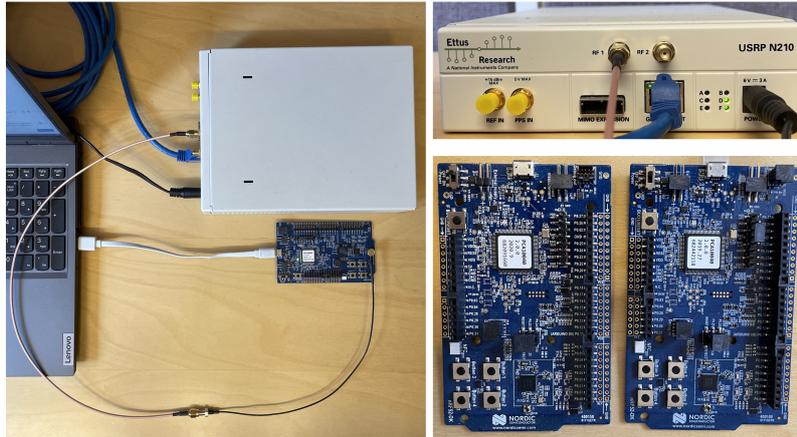


Figure 4: Equipment for acquiring amplitude-modulated EM emissions.

The target device is an nRF52832 chip mounted on a Nordic Semiconductors nRF52 DK development board. The chip supports Bluetooth 5 with a data transmission rate of 2Mbps. The option nRF5_SDK_14.2.0_17b948a is used for the radio setup.

The 32-bits ARM Cortex-M4 CPU contained in nRF52832 is programmed to the C implementation of SABER from [BDK⁺21] without any countermeasures against power/EM analysis. The C implementation is using gcc-arm-none-eabi-8-2018-q4-major with two different optimization options: -O0 (no optimization) and -O3 (highest level of optimization). The CPU runs at 64MHz.

The receiver is an Ettus Research USRP N210 software defined radio (SDR). The center receiving frequency is set to $2f_{clock} + f_{Bluetooth} = 2.528\text{GHz}$, where $f_{Bluetooth} = 2.4\text{GHz}$ is the Bluetooth channel center frequency and $f_{clock} = 64\text{MHz}$ is the frequency of the CPU clock.

The signals are sampled with the sampling frequency 25MHz, which is the maximum sampling frequency of USRP N210 SDR 25MHz, limited by interface. The signals are transmitted from the target device to the receiver through an SMA coaxial cable.

4.2 Trace pre-processing

Amplitude-modulated EM emanations are very noisy and thus need to be pre-processed to increase the signal-to-noise (SNR) ratio. Similarly to [WND22], we pre-process all traces by averaging 100 repeated measurements. This improves the SNR by a factor of ten, $\sqrt{100} = 10$.

In our experiments, we carry out the attacks on both, the profiling device and a different device, shown in Fig. 4. To reduce the negative effect of device intra-variability in the latter case, we apply two scaling methods: *min-max scaling* and *standardization* (also known as *variance scaling*) [ZC18].

Let \mathbb{R} denote the set of real numbers. Given a set of traces \mathcal{T} with elements of type $\mathcal{T} = (\tau_1, \dots, \tau_w) \in \mathbb{R}^w$, each trace $\mathcal{T} \in \mathcal{T}$ is scaled to $\mathcal{T}' = (\tau'_1, \dots, \tau'_w) \in \mathbb{R}^w$ such that

$$\tau'_i = \begin{cases} \frac{\tau_i - \tau_{min}}{\tau_{min} - \tau_{max}}, & \text{for min-max scaling} \\ \frac{\tau_i - \mu_i}{\sigma_i}, & \text{for standardization,} \end{cases}$$

where τ_{min} and τ_{max} are the minimum and the maximum data points in \mathcal{T} , and μ_i and σ_i are the mean and standard deviation of traces in \mathcal{T} at the i th trace point, for all $i \in \{1, \dots, w\}$.

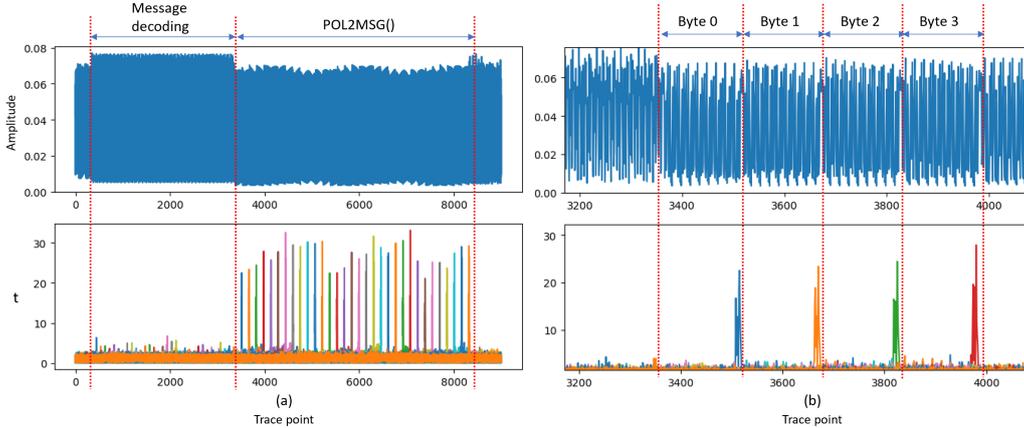


Figure 5: (a) A trace representing the message decoding and POL2MSG() procedures (top) and t-test for 32 message bytes (bottom) for 30K traces; (b) A zoomed-in view of the first four message bytes in POL2MSG() (top) and their t-test (bottom).

4.3 Selecting intervals of interest

To exploit the Byte-wise-Storage vulnerability in POL2MSG() procedure, we first locate the part of traces representing the execution of POL2MSG() during the decapsulation of the message. The message is decapsulated at the step 1 of Saber.KEM.Decaps() in Fig. 2, when the ciphertext c is decrypted by Saber.PKE.Dec().

According to the C implementation of POL2MSG() in Fig. 3, we expect to see 32 similarly looking patterns representing the packing of each block of eight message bits into a byte. The top part of Fig. 5(a) shows a segment of trace containing both, message decoding and POL2MSG() procedures. The top part of Fig. 5(b) shows a zoomed-in view of the first four message bytes in POL2MSG().

Once the approximate position of POL2MSG() is determined, we apply test vector leakage assessment (TVLA) to locate the intervals corresponding to the processing of each message byte more precisely. These intervals are used for training of the neural networks at the profiling stage.

The TVLA [GJJR11] is a well-known statistical method for evaluating side-channel leakage qualitatively. It applies Welch's t-test [Wel47] to compare the means of two sets of measurements, \mathcal{T}_0 and \mathcal{T}_1 :

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{\sigma_0^2}{n_0} + \frac{\sigma_1^2}{n_1}}},$$

where μ_i , σ_i and n_i are the mean, standard deviation and cardinality of the set \mathcal{T}_i , for $i \in \{0, 1\}$. The null hypothesis (that the difference in means is zero) is rejected with a confidence of 99.9999% if the absolute value of the t-test score is higher than 4.5 [Wel47]. This means that \mathcal{T}_0 and \mathcal{T}_1 have noticeable differences and hence may leak some side-channel information.

The bottom part of Fig. 5(a) shows t-test results for all 32 message bytes corresponding to the trace segment in the above plot. The t-test was carried out on a set of 30K traces captured for random messages and random keys. Each trace in the set is an average of 100 repeated measurements. We can clearly see 32 peaks in POL2MSG() part of the trace. In the message decoding part, the leakage is much weaker.

In the zoomed-in view of POL2MSG() leakage in Fig. 5(b) bottom, we can see that the

Table 1: The MLP architecture; $w = 160$ (20) for -O0 (-O3).

Layer (Type)	Output Shape	Parameter #
Input	w	0
BatchNormalization1	w	$4w$
Dense1 (ReLU)	512	82432
Dense2 (ReLU)	256	131328
Dense3 (ReLU)	256	65792
Dropout1	256	0
Output (Softmax)	256	65792

t-test peaks are located at the end of the corresponding byte processing. This is reasonable since, according to the implementation of `POL2MSG()` (see line 4 of `POL2MSG()` in Fig. 3), the packed byte value is stored in memory after the inner `for`-loop is completed.

5 Profiling stage

This section describes how we train neural networks at the profiling stage.

Let \mathbb{I} denote the set of real numbers within the interval $[0,1]$, $\mathbb{I} := \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$.

We use w -point segments of traces containing the execution of the i th message byte by `POL2MSG()` to train neural networks of type $\mathcal{N}_i : \mathbb{R}^w \rightarrow \mathbb{I}^{256}$ which predict the value of the i th message byte, for all $i \in \{0, 1, \dots, 31\}$. The set of training traces, \mathcal{T}_T , is captured for random messages and random keys. Message byte values are used as labels for traces.

Table 1 shows the architecture of neural networks in our experiments. The input size of the network is $w = 160$ and $w = 20$ points for the Saber implementation compiled with -O0 and -O3 optimization levels, respectively.

During training, we use *Nadam* optimizer with the learning rate of 0.0001 and numerical stability constant *epsilon*=1e-8. Categorical cross-entropy is used as a loss function to evaluate the network classification error. The number of epoch is set to 100 with a batch size 128. The dropout rate is set to 0.2. 10% traces are used for validation. Only the model with the highest validation accuracy is saved.

6 Attack stage

In this section, we present the new message recovery method based on multiple-bit error injection and describe how we use it to obtain Saber’s secret and session keys.

6.1 Multiple-bit error (MBE) method for message recovery

Let $m = (m[0], m[1], \dots, m[31])$ be a message of Saber KEM to be recovered, where $m[i]$ is the i th message byte, and $c = (\mathbf{c}_m, \mathbf{b}')$ be a properly generated ciphertext which contains m .

We create 255 modified versions of c , denoted by c_e , such that `Saber.PKE.Dec()` decrypts c_e to

$$m_e = (m[0] \oplus e, m[1] \oplus e, \dots, m[31] \oplus e), \quad (1)$$

where $e \in \{1, 2, \dots, 255\}$ is the error. The same error is injected into all message bytes in parallel. The original ciphertext c corresponds to the error-free case, $c = c_0$.

The modified ciphertexts c_e are created by changing the coefficients of \mathbf{c}_m so that, for every message byte $i \in \{0, 1, \dots, 31\}$, all bits of $m[i]$ in which the 8-bit binary expansion of e has the value 1 are flipped. To flip a message bit j , the value of the center of the integer

ring \mathbb{Z}_q is subtracted from the j th coefficient of \mathbf{c}_m , for any $j \in \{0, 1, \dots, 255\}$. Since the message polynomial is only additively hidden within \mathbf{c}_m (see line 8 of `Saber.PKE.Enc()`), this results in a ciphertext decrypting to a message equal to the original message m with the j th bit flipped [RBRC21].

Next we acquire 256 attack traces $\mathcal{T}_A = \{\mathcal{T}_0, \mathcal{T}_1, \dots, \mathcal{T}_{255}\}$ captured during the decapsulation of the ciphertext c_e by the device under attack, for all $e \in \{0, 1, \dots, 255\}$. For each message byte $i \in \{0, 1, \dots, 31\}$, the w -point segments containing the execution of $m[i]$ by `POL2MSG()` are located in \mathcal{T}_A and extracted. The extracted trace segments are given as input to the MLP model \mathcal{N}_i trained at the profiling stage.

For each $\mathcal{T}_e \in \mathcal{T}_A$, the model \mathcal{N}_i outputs a score vector $S_{i,e} = \mathcal{N}_i(\mathcal{T}_e)$ in which the value of the l th element, $S_{i,e}[l]$, is the probability that $m_e[i] = l$ in \mathcal{T}_e , for $l, e \in \{0, \dots, 255\}$.

The most likely label for $m[i]$ among 256 candidates is decided as:

$$\tilde{l} = \arg \max_{l \in \{0, 1, \dots, 255\}} \left(\prod_{e=0}^{255} S_{i,e}[l \oplus e] \right).$$

If $\tilde{l} = m[i]$, the classification is successful. The condition $\tilde{l} = m[i]$ can be verified by checking if the rank of the message byte i , $rank_i$, is zero.

Since we inject all possible multiple-bit errors into each message byte $i \in \{0, 1, \dots, 31\}$, for every i , the ground truth labels of 256 traces of \mathcal{T}_A are mutually disjoint. Therefore, at least one of the traces of \mathcal{T}_A has the label preferred by the model \mathcal{N}_i for every i .

6.2 Secret key recovery

The secret key is recovered as follows:

1. Use the method from [NDGJ21] described later in this section to construct 24 chosen ciphertexts c_1, \dots, c_{24} .
2. For each c_i , $i \in \{1, \dots, 24\}$, use the multiple-bit error injection method to construct 255 modified versions of c_i which decrypt to messages defined by eq. (1)
3. Capture from the device under attack a set of attack traces for 24×256 resulting ciphertexts.
4. Use the attack traces to recover the messages m_1, \dots, m_{24} contained in the ciphertexts c_1, \dots, c_{24} using the message recovery algorithm described in Section 6.1.
5. Derive the secret key \mathbf{s} from 24 recovered messages m_1, \dots, m_{24} as described below.

Following the method [NDGJ21], the ciphertexts are constructed as $c_i = (\mathbf{c}_m, \mathbf{b}')$ where $\mathbf{c}_m = k_0 \sum_{j=0}^{255} x^j \in R_T$ and

$$\mathbf{b}' = \begin{cases} (k_1, 0, 0) \in R_p^{3 \times 1} & \text{for } i = \{1, \dots, 8\}, \\ (0, k_1, 0) \in R_p^{3 \times 1} & \text{for } i = \{9, \dots, 16\}, \\ (0, 0, k_1) \in R_p^{3 \times 1} & \text{for } i = \{17, \dots, 24\}, \end{cases}$$

where the pairs (k_0, k_1) are defined in Table 2. The coefficients of the secret key \mathbf{s} are mapped into codewords of the $[8, 4, 4]_2$ extended Hamming code composed from the bits of eight messages. The first group of 256 secret key coefficients is derived from messages recovered from c_1, \dots, c_8 , the second group of 256 coefficients - from from messages recovered from c_9, \dots, c_{16} , and the last group of 256 coefficients - from messages recovered from c_{17}, \dots, c_{24} .

Table 2: The mapping of bits of eight messages into secret key coefficients [NDGJ21].

Coef. of s	The message bit value for the pair (k_1, k_0)							
	(186,0)	(293,7)	(311,7)	(615,2)	(613,2)	(890,4)	(903,4)	(199,0)
-4	0	1	1	1	1	0	0	0
-3	1	1	1	0	0	0	0	1
-2	1	0	0	1	1	0	0	1
-1	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	1	0
1	0	0	0	1	1	1	1	0
2	1	0	0	0	0	1	1	1
3	1	1	1	1	1	1	1	1
4	1	1	0	1	0	0	1	0

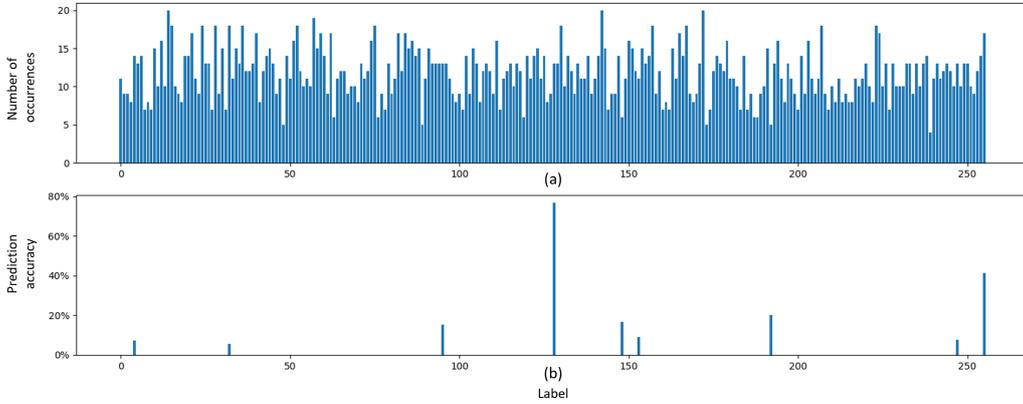


Figure 6: The distribution of (a) ground truth labels and (b) labels predicted by the model \mathcal{N}_0 trained on a 30K set. Results of a single-trace attack on the same device (the average of 3K traces captured for random messages).

6.3 Session key recovery

Given a properly generated ciphertext c , the session key can be trivially extracted by first recovering the message m contained in c from 256 traces using the presented multiple-error injection method. Then, the session key is computed as $K = \mathcal{H}(\hat{K}', c)$ where $(\hat{K}', r') = \mathcal{G}(pkh, m)$ (see lines 2 and 4 of Saber.KEM.Encaps()).

7 Experimental results

In the experiments, we use two identical nRF5283 devices, D_1 and D_2 , shown in Fig. 4. D_1 is used for capturing training traces for the profiling stage. Both D_1 and D_2 are used for capturing test traces for the attack stage. All training and test traces are pre-processed by averaging 100 repeated measurements.

The experiments in Sections 7.1-7.3 are carried out on devices programmed to the Saber implementation compiled with -O0 optimization level. In Section 7.4, we show results for the implementation compiled with -O3 optimization level.

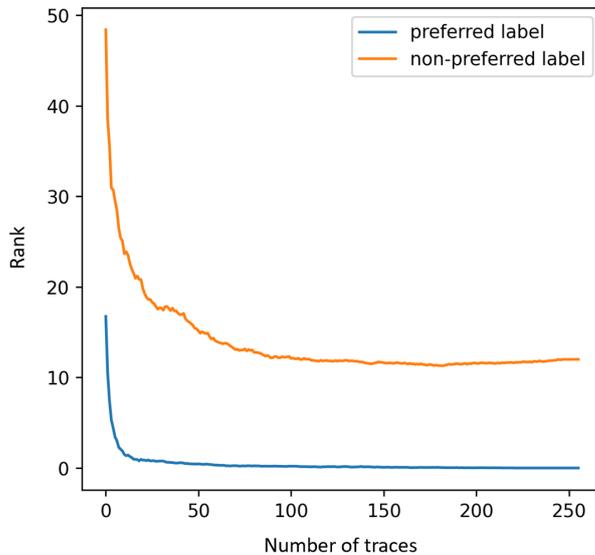


Figure 7: The rank of $m[0]$ in a repetition attack using 256 traces with the same ciphertext.

7.1 Bias in neural networks

In this section, we demonstrate that multi-class neural networks which are trained on a small dataset may be strongly biased towards certain classes in their predictions. This phenomenon has been observed in previous side-channel attacks, e.g. [BF19].

We trained an MLP model \mathcal{N}_0 with the architecture listed in Table 1 on 30K traces captured for random messages (with 10% left for validation). The model was trained on the segment of `POL2MSG()` corresponding to the processing of the first message byte, $m[0]$.

After training, we tested \mathcal{N}_0 on 3K traces from the same device captured for random messages. Each prediction was done based on a single trace (single-trace attack). Fig. 6 illustrates the results. The top plot shows the distribution of ground truth labels in the 3K attack set. We can see that the labels are more or less uniform. The bottom plot shows the distribution of labels predicted by \mathcal{N}_0 . There is a strong bias towards one label, 128, which is predicted correctly with 75% probability. In the rest of the section, we call such labels *preferred*. We can also see that, the majority of labels, 96.5%, are predicted with 0% probability. We refer to them as *non-preferred*.

We believe that the strong bias of \mathcal{N}_0 is due to the fact that the 256-class model was trained on a small dataset in which each class appears only roughly 100 times. This does not seem sufficient. For a comparison, in the single-trace attack on Saber presented in [NDGJ21], using power side-channels, a 1.6M dataset was used for training 2-class MLP models which achieve 0.997% message bit prediction accuracy. In their training dataset, each class from $\{0, 1\}$ appears 0.8M times. This is four orders of magnitude larger compared to the number of occurrences of each class from $\{0, 1, \dots, 255\}$ in the 30K dataset in our experiment. Another reason can be that MLP models are quite simple. More complex DL models, e.g. transformers [VSP⁺17], may achieve better results [Bri21].

The key idea of the presented multiple-bit error injection method is that, instead of increasing the training set by several orders of magnitude to get unbiased models, we increase the attack set 256 times and achieve high prediction accuracy with biased models. Since the the attack set is several orders of magnitude smaller than the training set, the presented method minimizes the total number of training plus attack traces required for a successful attack.

One can ask if a similar improvement in the success rate can be achieved by a repetition

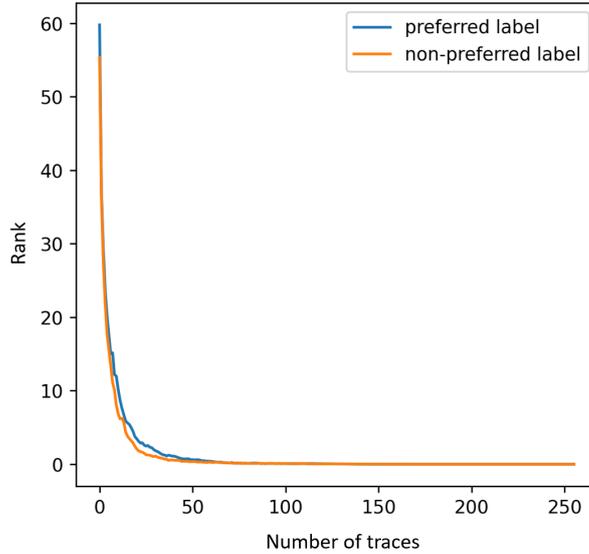


Figure 8: The rank of $m[0]$ in an attack using 256 traces of MBE method.

attack which uses 256 traces captured for the *same* ciphertext c . Fig. 7 and 8 show that the repetition method is not as good as the multiple-bit error injection. For both methods, we used the model \mathcal{N}_0 from the previous experiment for predicting labels in two scenarios:

1. The ground truth label of a trace in the attack set captured with c as input is a preferred label of \mathcal{N}_0 (blue plot).
2. The ground truth label of a trace in the attack set captured with c as input is a non-preferred label of \mathcal{N}_0 (orange plot).

From Fig. 7 we can see that, in the repetition attack, \mathcal{N}_0 successfully recovers the former and fails to recover the latter. Contrary, Fig. 8 shows that, in the attack using multiple-bit error injection method, \mathcal{N}_0 successfully recovers labels in both cases. This is not surprising since the injected errors assure that at least one of the 256 traces in the attack set has a label preferred by the model.

7.2 Message recovery attack

In this section, we evaluate how scaling, ensemble learning and repetitions affect the success rate of a message recovery attack based on the multiple-bit error injection method.

7.2.1 Scaling

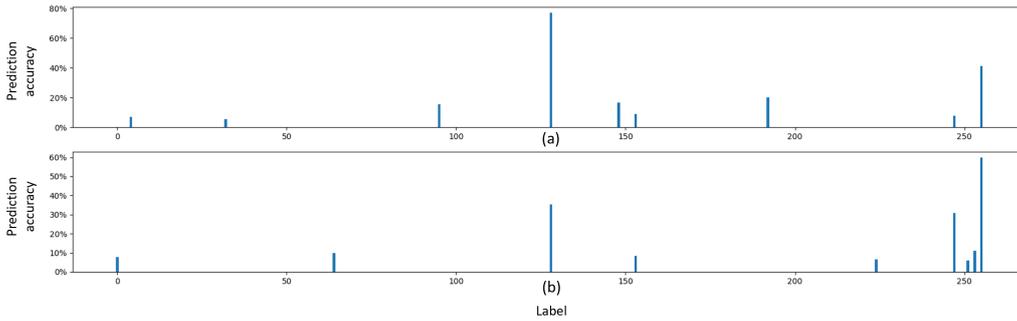
First, we evaluate the impact of different scaling methods on the probability of message recovery from a different second device.

At the profiling stage, we captured from the profiling device, D_1 , a set of 30K traces for random messages, \mathcal{T}_T . Then we scaled \mathcal{T}_T using two different methods: min-max normalization and standardization. Using the profiling strategy described in Section 5, for each message byte $i \in \{0, 1, \dots, 31\}$, we trained models \mathcal{N}_i on each of these three training sets.

At the attack stage, we selected at random five different messages and computed the corresponding ciphertexts using the public key of the device under attack, D_2 . These five ciphertexts, together with their 255 multiple-bit error injected versions, were applied as

Table 3: The impact of scaling on the average empirical probability of recovering a message byte from 256 traces of MBE method captured from a different device.

Scaling method	Message					Average
	1	2	3	4	5	
No scaling	0.6652	0.8135	0.6865	0.6647	0.7891	0.7238
Min-max normalization	0.7649	0.8412	0.7781	0.7315	0.8576	0.7947
Standardization	0.8958	0.9271	0.8698	0.9167	0.9323	0.9083

**Figure 9:** The preferred labels of two models trained on the same 30K set with a different order of elements. Results of a single-trace attack on the same device (the average of 3K traces for random messages).

inputs to D_2 to capture the set of attack traces, \mathcal{T}_A . The set \mathcal{T}_A was scaled using the same two methods as \mathcal{T}_T .

Table 3 lists the average empirical probabilities of recovering a message byte from \mathcal{T}_A for each of the five messages. We calculate the probabilities as $p_i = \frac{1}{1+rank_i}$, where $i \in \{0, 1, \dots, 31\}$ is the byte number.

We can see that both min-max normalization and standardization scaling methods improve the message byte recovery probability. For the standardization, the average probability is by 18.45% larger than the one for non-scaled traces. In the rest of experiments, we use traces scaled with the standardization method.

7.2.2 Ensemble learning

Next, we evaluate if the probability of message recovery can be further improved by using an ensemble of models. It is known that, if the models make independent errors, the ensemble can perform considerably better than its members [GBC16].

Since we randomly shuffle traces in a training set for each training session and set aside 10% of the set for validation, at each training session the models are trained on a slightly different set. In addition, data in the beginning of the training set seem to have a higher impact on the model than the data at the end. Due to these and other factors, two models trained on the same dataset may have different preferred labels, as illustrated in Fig. 9. It shows the results of a single-trace attack for two models trained on the same 30K set. We can see that some of their preferred labels are different. This implies that the models may be making different errors on the same attack set and, hence, the ensemble approach might be beneficial.

To verify the latter, we used the same 30K training set to train 10 different models for each message byte $i \in \{0, 1, \dots, 31\}$ and used an ensemble of k of these models to recover the bytes using the multiple-bit error injection method. Table 4 summarizes the results.

Table 4: The average empirical probability of recovering a message byte from 256 traces of MBE method captured from a different device using an ensemble of k models.

Number of models, k	1	2	3	4	5
Average probability	0.9083	0.9370	0.9469	0.9573	0.9458
Number of models, k	6	7	8	9	10
Averaged probability	0.9385	0.9448	0.9510	0.9510	0.9417

Table 5: The average empirical probability of recovering a message byte and the complete message from $256 \times N$ traces of MBE method captured from a different device.

	Message					Average	Complete message
	1	2	3	4	5		
$N = 1$	0.9688	0.9844	0.9323	0.9375	0.9635	0.9573	0.2475
$N = 2$	0.9844	0.9688	1.0000	0.9531	1.0000	0.9813	0.5459
$N = 3$	1.0000	0.9688	1.0000	0.9844	1.0000	0.9906	0.7401

Table 6: The average empirical probability of recovering a message byte and the complete message from $256 \times N$ traces of MBE method captured from the same device.

	Message					Average	Complete message
	1	2	3	4	5		
$N = 1$	0.9688	1.0000	1.0000	0.9844	0.9844	0.9875	0.6691
$N = 2$	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
$N = 3$	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

We can see that combining four models into an ensemble is the best choice. In the rest of experiments, we use an ensemble of four models.

7.2.3 Repetitions

Finally, we investigate if the probability of message recovery can be further improved if each trace in the attack set is captured with N repetitions.

Table 5 shows the results for $N = 1, 2$ and 3 for the case when the device under attack is different from the profiling device. We can see that, by raising the degree of repetition N to 3 , we can boost the average probability of recovering a message byte to 0.9906 and hence the likelihood of recovering the complete message to 0.7401 . We believe that, by raising N , the latter can be further improved.

Table 6 presents similar results for the case when the device under attack is the same as the profiling device. We can see that, in this case, the probability of recovering the message is 1 for $N \geq 2$. We show both tables to emphasize the significant impact of device intra-variability and justify the advantage of profiling on the device under attack (and hence the need for minimizing the training set).

7.3 Secret key recovery attack

It follows from Tables 5 and 6 that we may recover messages with some errors. For this reason, for secret key recovery, we use the ECC-based approach for constructing chosen ciphertexts presented in [NDGJ21] rather than the methods from [RBRC21, RSRCB20]. The methods [RBRC21, RSRCB20] use half as many ciphertexts as the method [NDGJ21] to

Table 7: The statistic on different types of errors in a secret key recovery attack using $24 \times 256 \times N$ traces of MBE method from a different device.

N	Correct predictions		Errors	
	No errors	Errors corrected by ECC	Detected errors	Undetected errors
1	665	84	17	2
2	744	22	2	0
3	758	10	0	0
4	766	2	0	0
5	766	2	0	0

recover the key. However, they cannot correct any error. The ECC approach of [NDGJ21] can correct 1-bit error in every coefficient of the secret key \mathbf{s} .

The secret key recovery attack follows the steps (1)-(5) described in Section 6.2. First, 24 messages m_1, \dots, m_{24} contained in the chosen ciphertexts c_1, \dots, c_{24} are recovered using the message recovery attack based on the multiple-bit error injection method. These messages are used to derive 768 coefficients of secret key \mathbf{s} based on the mapping in Table 2.

To evaluate the attack, we group possible outcomes into four cases:

1. *No errors*: The recovered coefficient matches the ground truth key coefficient.
2. *Errors corrected by the ECC*: There is exactly one error in the eight message bits. This error is corrected by the ECC.
3. *Detected errors*: ECC detects more than one errors in the eight message bits and this combination of bits is not in Table 2. These errors are detected by the ECC.
4. *Undetected errors*: The combination of the eight message bits is in Table 2, but the recovered coefficient does not match the ground truth secret key coefficient.

The case (4) implies a failed secret key recovery because any number of wrong coefficients makes the recovered key useless. The errors in case (3) may be fixed by key enumeration if their number is small, since the location of the error is known. The complexity of key enumeration is 9^n , where n is the number of detected errors. Table 7 list the statistic on the number of occurrences of each of the four cases for different degrees of repetitions N .

For $N = 1$, there are 2 undetected errors, so the attack fails. For $N = 2$, there are no undetected errors and only 81 enumerations are required to find the secret key. Therefore, the attack is successful. For $N \geq 3$, the secret key can be recovered without any key enumeration.

7.4 Higher optimization level

All experiments in the previous sections are made on the devices programmed with the implementation of Saber compiled with `-O0` optimization level. In this section, we present the results for the implementation compiled with the highest optimization level, `-O3`.

First, we compared side-channel leakage in both implementations. Table 8 lists the maximum t-test score for all message bytes. We can see that, on average, the leakage from `-O3` is 4.3 times weaker than the one from `-O0`.

Next, we trained models \mathcal{N}_i for each message byte $i \in \{0, 1, \dots, 31\}$ using same size of the training set, 30K, captured from D_1 and the same training strategy as in the experiments with `-O0` optimization level. Fig. 10 shows the ranks for all message bytes recovered using an ensemble of four models from $256 \times N$ traces with the degree of repetition $N = 15$ captured from a different device, D_2 .

Table 8: Maximum t-test scores for all message bytes in -00 and -03 implementations.

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-00	22.5	23.4	24.4	27.9	25.7	27.7	32.6	29.5	29.1	30.2	29.8	30.4	22.4	22.5	23.7	27.6
-03	7.2	5.3	7.2	6.1	6.4	4.3	7.1	5.9	6.2	3.8	6.4	5.6	6.6	5.8	7.0	7.1

Byte	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	Avg
-00	26.0	27.2	31.6	28.8	27.5	29.9	30.6	33.1	25.5	21.1	25.1	25.1	23.7	27.4	29.0	29.2	27.2
-03	5.8	5.6	6.2	6.0	6.8	6.3	6.4	7.9	6.8	5.7	6.9	6.2	6.2	5.3	5.1	8.9	6.3

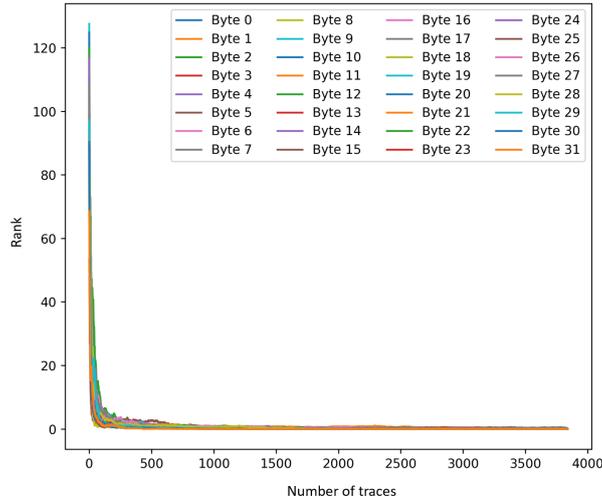


Figure 10: The rank of $m[i]$ in an attack by an ensemble of four models $\mathcal{N}_i, i \in \{0, 1, \dots, 31\}$, based on $256 \times N$ traces of MBE method from a different device for $N = 15$.

Even though the leakage from -03 implementations is much weaker than the one from -00, the ranks of all bytes reach 0 with a higher degree of repetition $N = 15$. Therefore, the message recovery attack successfully recovers a complete message from traces captured from a different device.

We have not tried to recover the secret key from the implementation complied with -03 optimization level, but there should be no principle difference from -00 case since the message recovery attack for -03 optimization level is successful.

8 Conclusion

We presented the first side-channel attack which can successfully recover messages of Saber KEM from its software implementation using amplitude-modulated EM emanations. Previous amplitude-modulated EM emanation-based attacks on Saber KEM were not able to recover complete messages with a sufficiently high probability. We also demonstrate a successful secret key recovery attack on Saber KEM from messages extracted for chosen ciphertexts.

The presented multiple-bit error injection method is not specific for Saber KEM. It can be applied to other LWE/LWR-based PKE/KEMs, or, more generally, to any cryptographic algorithm in which a secret can be manipulated by modifying input data controlled by the attacker. It is also applicable to any type of side-channels, but seems to be most valuable in situations where the leakage is weak.

9 Acknowledgments

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

- [BCL⁺17] Daniel J Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, et al. *Classic mceliece*. 2017.
- [BDK⁺21] Michiel Van Beirendonck, Jan-Pieter D’anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel-resistant implementation of Saber. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 17(2):1–26, 2021.
- [BF19] Martin Brisfors and Sebastian Forsmark. Deep learning side-channel attacks on AES. Bachelor’s thesis, School of Electrical Engineering and Computer Science, KTH, 2019.
- [Bri21] Martin Brisfors. Advanced side-channel analysis of USIMs, Bluetooth SoCs and MCUs. Master’s thesis, School of Electrical Engineering and Computer Science, KTH, 2021.
- [C⁺20] C. Chen et al. NTRU algorithm specifications and supporting documentation. 2020.
- [CFS20] Giovanni Camurati, Aurélien Francillon, and François-Xavier Standaert. Understanding screaming channels: From a detailed analysis to improved attacks. *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pages 358–401, 2020.
- [CPM⁺18] Giovanni Camurati, Sebastian Poehlau, Marius Muench, Tom Hayes, and Aurélien Francillon. Screaming channels: When electromagnetic side channels meet radio transceivers. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 163–177, 2018.
- [D⁺20] J. D’Anvers et al. Saber algorithm specifications and supporting documentation. 2020.
- [FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Annual international cryptology conference*, pages 537–554. Springer, 1999.
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. In *NIST Non-Invasive Attack Testing Workshop*, 2011.
- [GJN20] Qian Guo, Thomas Johansson, and Alexander 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*, pages 359–386. Springer, 2020.

- [MBB⁺22] Catinca Mujdei, Arthur Beckers, Jose Bermundo, Angshuman Karmakar, Lennert Wouters, and Ingrid Verbauwhede. Side-channel analysis of lattice-based post-quantum cryptography: Exploiting polynomial multiplication. *Cryptology ePrint Archive*, 2022.
- [NDGJ21] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked IND-CCA secure Saber KEM implementation. *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pages 676–707, 2021.
- [NDJ21] Kalle Ngo, Elena Dubrova, and Thomas 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*, pages 51–61, 2021.
- [NIS16] 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>.
- [RBRC21] Prasanna Ravi, Shivam Bhasin, Sujoy Sinha Roy, and Anupam Chattopadhyay. On exploiting message leakage in (few) NIST PQC candidates for practical message recovery attacks. *IEEE Transactions on Information Forensics and Security*, 2021.
- [RSRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. 2020:307–335, Jun. 2020.
- [S⁺20] P. Schwabe et al. CRYSTALS-Kyber algorithm specifications and supporting documentation. <https://csrc.nist.gov/projects/postquantum-cryptography/round-3-submissions>, 2020.
- [SKL⁺20] 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.
- [UXT⁺22] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/em analysis on post-quantum KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 296–322, 2022.
- [VSP⁺17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.
- [Wel47] Bernard L Welch. The generalization of ‘student’s’ problem when several different population variances are involved. *Biometrika*, 34(1-2):28–35, 1947.
- [WND22] Ruize Wang, Kalle Ngo, and Elena Dubrova. Side-channel analysis of saber kem using amplitude-modulated em emanations. *Cryptology ePrint Archive*, Paper 2022/807, 2022.
- [ZC18] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. " O’Reilly Media, Inc.", 2018.