Curse of Re-encryption: A Generic Power/EM Analysis on Post-Quantum KEMs

Rei Ueno[1,2,3], Keita Xagawa[4], Yutaro Tanaka[1,2], Akira Ito[1,2], Junko Takahashi[4] and Naofumi Homma[1,2]

[1] Tohoku University, 2–1–1 Katahira, Aoba-ku, Sendai-shi, 980-8577, Japan
rei.ueno.a8@tohoku.ac.jp, {y-tanaka,ito,homma}@riec.tohoku.ac.jp
[2] CREST, JST, 4–1–8 Honcho, Kawaguchi, Saitama, 332-0012, Japan
[3] PRESTO, JST, 4–1–8 Honcho, Kawaguchi, Saitama, 332-0012, Japan
[4] NTT Secure Platform Laboratories, Nippon Telegraph and Telephone Corporation, 3–9–11 Midori-cho, Musashino-shi, Tokyo, 180-8535, Japan
keita.xagawa.zv@hco.ntt.co.jp, junko.takahashi.fc@hco.ntt.co.jp

Abstract. This paper presents a side-channel analysis (SCA) on key encapsulation mechanism (KEM) based on Fujisaki–Okamoto (FO) transformation. FO transformation has been widely used for realizing actively secure KEMs from passively secure public key encryption (PKE), as it is employed in most of NIST post-quantum cryptography (PQC) candidates for KEM. The proposed attack exploits side-channel leakage from target implementation during execution of psuedorandom function (PRF) in re-encryption of KEM decapsulation as a *plaintext-checking oracle* that tells whether the PKE decryption result is equivalent to the *reference* plaintext. Due to the generality and practicality of the plaintext-checking oracle, the proposed attack can attain a full key recovery of various KEMs where an active attack on the underlying PKE is known. This paper demonstrates that the proposed attack can perform a full key recovery on most NIST PQC third-round candidates for KEM, namely, Kyber, Saber, FrodoKEM, NTRU Prime, NTRU, HQC, BIKE, and SIKE. For BIKE, the proposed attack achieves a partial key recovery. The applicability to Classic McEliece is unclear because no active attack on Classic McEliece using a plaintext-checking oracle is known. This paper also presents a side-channel distinguisher design based on deep learning (DL) for mounting the proposed attack on practical implementation, which requires no profiling device. This paper validates the attack feasibility through experimental attacks on various PRF implementations (herein, a SHAKE software, an AES software, a bit-sliced masked AES software, and a masked AES hardware based on threshold implementation). As a result, we confirm the practicality as the proposed SCA is successful for all implementation used in the experiment.

Keywords: Side-channel analysis · Fujisaki–Okamoto transformation · Key encapsulation mechanism · Public key encryption · Post-quantum cryptography · Deep learning

1 Introduction

1.1 Background

Public key encryption (PKE) is a cryptographic primitive essential for secure information systems based on secret key exchange, digital signature, *etc.* Since it is usually difficult to construct a chosen ciphertext attack (CCA)-secure PKE, the Fujisaki–Okamoto (FO) transformation [FO99] and its variant (*e.g.*, [HHK17, SXY18, BHH+19]) have been commonly used to realize CCA-secure key encapsulation mechanisms (KEMs) from a chosen

plaintext attack (CPA)-secure PKE on the basis of *re-encryption*, as for most of KEM schemes in NIST post-quantum cryptography (PQC) competition [NIS20]. Although the theoretical/mathematical security of such KEM schemes has been severely analyzed, side-channel analysis (SCA), which is a type of attack on cryptographic implementation using side-channel leakage (*e.g.*, execution time, power consumption, and electromagnetic (EM) emanation), may break these schemes when they are implemented in the real world [Koc96, KJJ99]. It is quite important to investigate the SCA vulnerability of KEM schemes for the applications in which SCA can be a practical threat, such as Internet-of-Things (IoT).

Many previous SCAs on KEM have mainly focused on the decryption of the underlying PKE in order to recover the secret key (*e.g.*, [PPM17, ZYD+20]). In contrast, recently, some studies show that the implementation of FO transformation may leak secret key, even if the underlying PKE is securely implemented [GTN20, RRCB20, PP21]. These attacks exploit side-channel leakage or fault injection to obtain the information about the PKE decryption result, and then mount a chosen-ciphertext attack on the underlying PKE. In [GTN20], Guo *et al.* presented a timing attack that is potentially applicable to lattice- and code-based KEMs if the FO transformation is implemented in a non-constant-time manner, which reveals the importance of constant-time implementation of FO transformation in addition to PKE. By contrast, for the power/EM side-channel and fault injection, attacks only applicable to lattice-based schemes are known [RRCB20, PP21]. In particular, there is no known power/EM SCA on FO transformation in code- and isogeny-based KEM(s) (*e.g.*, HQC, BIKE, and SIKE). The detailed evaluation of applicability/limitation of SCAs on FO transformation is essential for developing an adequate countermeasure for the sake of secure KEM implementation.

1.2 Our contributions

In this paper, we show that the side-channel leakage of re-encryption, which plays an essential role to realize a CCA security for most schemes, can be *generally* exploited to break the CCA security. We also present a concrete and practical method to exploit the leakage with experimental evaluation. The contributions of this paper are listed below.

- We present a generic power/EM SCA methodology for KEMs based on FO transformation and its variant. The key idea of the proposed attack is to realize a *plaintext-checking oracle* through a side-channel trace to mount a chosen-ciphertext attack on the underlying CPA-secure PKE. The oracle tells whether or not the PKE decryption result in decapsulation is equivalent to the *reference* plaintext, which means the decryption result corresponding to the valid ciphertext. To realize the oracle, the proposed attack exploits side-channel leakage during execution of psuedo-random function (PRF) in re-encryption of KEM decapsulation for distinguishing whether or not the PKE decryption result is the fixed reference plaintext. Therefore, the proposed attack can be performed even if the underlying PKE implementation has no secrecy leakage. As many PKEs are known to be vulnerable to active attack using the plaintext-checking oracle, the proposed attack can be widely applied to many KEM implementations based on FO transformation including lattice-, code-, and isogeny-based ones. Note that, although an SCA on masked polynomial comparison was reported for a plaintext-checking oracle and its application to Kyber was shown in [BDH+21], its generality was not discussed (At most, the existing work discussed only lattice-based KEMs and mentioned only Kyber, Saber, and FrodoKEM).

- We investigate the applicability of the proposed attack to NIST PQC third-round candidates for KEMs (four finalists and five alternatives), and demonstrate that Kyber, Saber, FrodoKEM, NTRU Prime, NTRU, HQC, BIKE, and SIKE are vulnerable

Table 1: Applicability of implementation attack focusing on FO transformation to NIST PQC third-round candidates for KEM and their possible countermeasure

		[GTN20]	[PP21]	[RRCB20]	**This work**
	Attack type	Timing	Fault	Power/EM	
Lattice	Kyber	Yes	Yes	Yes	**Yes**
	Saber	Yes	Yes	Yes	**Yes**
	FrodoKEM	Yes	No	Yes	**Yes**
	NTRU Prime	Partially yes†	No	No	**Yes**
	NTRU	Yes	No	No	**Yes**
Code	HQC	Yes	No	No	**Yes**
	BIKE	Yes	No	No	**Yes***
	Classic McEliece	Unknown	No	No	**Unknown**
Isogeny	SIKE	No	No	No	**Yes**
	Countermeasure	Constant-time	Redundancy	Masking	**Unknown**

† Applicable to NTRU LPRime, but not to Streamlined NTRU Prime.
* Partial-key recovery, not full-key recovery.

to the proposed SCA. The proposed attack achieves a partial key recovery of BIKE. The applicability to Classic McEliece is unclear since no adaptive attack using the plaintext-checking oracle is known. The applicability of the proposed and conventional attacks are summarized in Table 1, which shows the generality of the proposed attack, even compared to the conventional attacks. We stress here that this paper is the first report on power/EM analysis on FO transformation of code- and isogeny-based KEMs, although some SCAs on FO transformation of lattice-based KEMs have been already known in the previous works (*e.g.*, [RRCB20] on Kyber, Saber, and FrodoKEM and [REB+21] on Streamlined NTRU Prime). In addition, in this paper, we also introduce some tricks to reduce the number of oracle accesses (*i.e.*, side-channel traces) for an efficient key recovery using a more sophisticated plaintext-checking oracle than the aforementioned simple plaintext-checking oracle.

- We present a deep-learning (DL)-based distinguisher for implementing the plaintext-checking oracle, which is designed for a two-classification neural network (NN), and allows us to perform an attack without specific assumption and knowledge about the target implementation. In addition, we also describe how to distinguish the input with a convincing accuracy using an NN model whose accuracy is insufficient, as the accuracy of an NN model for SCA may be low due to the presence of noise and/or SCA countermeasure. Thus, the proposed NN-based distinguisher can be adopted of attacking practical implementations in a black-box manner even with an SCA countermeasure such as masking, as demonstrated in this paper. Note that the proposed attack requires no profiling device for acquiring a training dataset since it is acquired from the target implementation in the scenario of proposed attack, as in several previous SCAs on lattice-based KEMs such as [XPRO20, RBRC20, SKL+20, NDGJ21].

- Using the distinguisher, we validate the proposed attack through experimental attacks on various PRF implementations. In the experiment, we targeted a non-protected SHAKE and AES software provided in an open-source cryptographic software library `pqm4` [KRSS19, pqm21], an open-source masked AES software for ARM Cortex-M4 corresponding to Schwabe's and Stoffelen's paper [SS16, git21], and a masked AES hardware based on threshold implementation (TI) in [UHA17] as TI is one of the most promising masking schemes. As a result, we confirm that the NN-based distinguisher can achieve almost 100% accuracy on non-protected implementation and it can achieve a meaningful accuracy for protected implementation. We then severely and comprehensively evaluate the number of side-channel traces required for a successful

Algorithm 1 CCA-secure KEM based on FO transformation (KeyGen, Encaps, Decaps)

Input: 1^λ
Output: $\mathsf{sk}, \mathsf{pk}, s$
1: **Function** KeyGen(1^λ)
2: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$;
3: $s \leftarrow_\$ \mathcal{M}$;
4: **return** $(\mathsf{sk}, \mathsf{pk}, s)$;
5: **end Function**

Input: pk
Output: c, k
1: **Function** Encaps(pk)
2: $m \leftarrow_\$ \mathcal{M}$;
3: $r \leftarrow G(m[, \mathsf{pk}])$;
4: $c \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m; r)$;
5: $k \leftarrow \mathsf{KDF}(m, c)$;
6: **return** (c, k);
7: **end Function**

Input: $c, \mathsf{sk}, \mathsf{pk}, s$
Output: k
1: **Function** Decaps($c, \mathsf{sk}, \mathsf{pk}, s$)
2: $m' \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$;
3: $r' \leftarrow G(m'[, \mathsf{pk}])$;
4: $c' \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m'; r')$;
5: **if** $c = c'$ **then**
6: **return** $\mathsf{KDF}(m, c)$;
7: **else**
8: **return** $\mathsf{KDF}(s, c)$;
9: **end if**
10: **end Function**

key recovery, which shows the practicality of the proposed SCA on the post-quantum KEMs.

1.3 Paper organization

The remainder of this paper is organized as follows. Section 2 reviews the KEM based on FO transformation and the previous SCAs on KEMs focusing on FO transformation. Here, we does not review the previous studies on fault analysis because they are not closely related to the proposed attack. Section 3 describes the proposed SCA methodology on the basis of a plaintext-checking oracle realized *via* side-channel leakage. Section 4 demonstrates its application to post-quantum KEMs in NIST PQC third-round candidates. Section 5 presents the side-channel distinguisher design for mounting the proposed attack on practical implementation and Section 6 conducts experimental validation using various PRF implementations. Finally, Section 7 concludes this paper.

2 Related Works

2.1 IND–CCA-secure KEM based on FO transformation

KEM is a public key cryptographic primitive that encapsulates a secret key. KEM is defined as a triple of polynomial-time algorithms: a key generation KeyGen, an encapsulation Encaps, and a decapsulation Decaps. Many CCA-secure KEMs are realized using a CPA-secure PKE with FO transfromation or its variant (*e.g.*, [HHK17, SXY18, BHH+19]), as seen in most of NIST PQC candidates for KEMs.

Algorithm 1 illustrates $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ based on a (standard) FO transformation with implicit rejection, where PKE is a CPA-secure probabilistic PKE composed of a key generation algorithm Gen, an encryption algorithm Enc, and a decryption algorithm Dec. Here, let us consider a KEM that returns a random number for the case of an invalid ciphertext instead of a rejection symbol $\perp$ for simplicity. In KEM.KeyGen, given a security parameter 1^λ, we first generate a key pair $(\mathsf{sk}, \mathsf{pk})$ using the PKE key generation PKE.Gen. We then generate s as a random plaintext of the PKE from the message space $\mathcal{M}$ at Line 3. Finally, the algorithm returns a triplet $(\mathsf{sk}, \mathsf{pk}, s)$.

In KEM.Encaps, we first randomly generate a message m from $\mathcal{M}$. We then evaluate a PRF G for m or a pair of m and pk (*e.g.*, in the cases for BIKE and SIKE, respectively). In Line 4, we perform the PKE encryption PKE.Enc using a public key pk, message m, and randomness r. After we derive the shared secret k using a key derivation function KDF on m and c, the algorithm returns the ciphertext c corresponding to k. Note here that the ciphertext c may be a tuple.

In KEM.Decaps, we first perform the PKE decryption for c using the secret key sk to obtain plaintext m'. Then, as same as KEM.Encaps, we generate r' as $G(m')$ or $G(m', \mathsf{pk})$,

and evaluate $\mathsf{PKE.Enc}(\mathsf{pk}, m'; r')$. This procedure is so-called *re-encryption*. Then, at Line 5, we examine whether the re-encryption result c' is equal to the ciphertext c. If $c = c'$, the algorithm returns the shared secret $k = \mathsf{KDF}(m, c)$ as the ciphertext is valid; otherwise, the algorithm returns a random number of $\mathsf{KDF}(s, c)$ (instead of $\perp$) as the ciphertext is invalid. Thus, the KEM scheme gives any active attacker no information about the PKE decryption result for invalid ciphertext.

In many modern KEM schemes, the PRF G and KDF are instantiated using AES or SHAKE (or SHA3). There are some variants of FO transformation for different types of CPA-secure PKE (*e.g.*, deterministic PKE), different security models, tighter security bounds, and/or improved efficiency (*e.g.*, [HHK17, SXY18, BHH+19]); however, note that the basic principle is almost the same (that is, it is related to PRF, re-encryption, or validity check). Although some CCA-secure transforms avoid the complete re-encryption for computational efficiency (*e.g.*, [DV21] and NTRU submitted to NIST PQC), the proposed SCA would be applicable to the variants of FO transformation as long as they employ PRF and/or procedure corresponding to validity check.

2.2 Previous SCAs on FO transformation

2.2.1 Timing analysis

In [GTN20], Guo *et al.* presented the first SCA focusing on FO transformation. The attack utilizes a timing side-channel to realize a plaintext-checking oracle for lattice- and code-based KEMs. Since the timing attack exploits the equality check between the ciphertext and re-encryption result (*i.e.*, Line 5 in KEM.Decaps of Algorithm 1) rather than PKE.Dec, the attack can be applied to constant-time PKE implementation, unless whole decapsulation is implemented in a constant-time manner.

More precisely, the timing attack is a chosen ciphertext attack on KEMs and utilizes a plaintext-checking oracle to mount an active attack on the underlying lattice- or code-based PKE. Let c be a valid ciphertext named reference ciphertext corresponding to a plaintext m. For an invalid ciphertext c', the plaintext-checking oracle tells whether or not the PKE decryption result of c' is equivalent to m. Using such an oracle, active attacks are known for many lattice- and code-based PKEs, as demonstrated in [GTN20].

In the timing attack, the attacker generates an invalid ciphertext $c' = c + \delta$ where δ is determined according to the active attack. For lattice- and code-based PKEs, if δ is so small for the underlying scheme, the PKE decryption result of c' is equivalent to m, which indicates that the re-encryption result should be c in this case. Otherwise, the PKE decryption result is a random plaintext $\hat{m}$, which is re-encrypted to a random ciphertext $\hat{c}$ far different from c. Then, PKE.Decaps compares the ciphertext $c + \delta$ and the re-encryption result. Here, ciphertext of lattice- and code-based PKEs is treated as a long vector in common processors. Therefore, if two ciphertexts are considerably similar to each other (*i.e.*, if comparing $c + \delta$ and c), a standard comparison method (*e.g.*, `memcmp`) takes a relatively long time; otherwise (*i.e.*, if comparing $c + \delta$ and $\hat{c}$), the comparison terminates immediately after examining the first block comparison. This results in a timing difference depending on whether or not the PKE decryption result is equivalent to m; thus, the timing side-channel acts as a plaintext-checking oracle. The full-key recovery of the KEM scheme is achieved by repeatedly using the plaintext-checking oracle for different δ's determined according to the PKE scheme.

Guo *et al.* demonstrated the application of this attack to FrodoKEM using a simulation in [GTN20]. Although the signal-to-noise ratio of side-channel measurement (*i.e.*, accuracy of the oracle) would be problematic, the result indicates that the full-key recovery would be sufficiently feasible. Due to the disclosure of this attack, many PQC implementations have employed a fully constant-time conditional move (*e.g.*, `cmov`) for realizing the comparison of c and c' and the move operation in PKE.Decaps. Thus, the timing attack is prevented

at this time.

Note that the timing attack cannot be applied to SIKE (the isogeny-based KEM in NIST PQC), because the known active attack on SIKE.PKE uses very different invalid ciphertext(s) from reference ciphertext, which indicates that the comparison operation between c and c' immediately terminates independently of whether the PKE decryption result is m or not. In addition, Guo *et al.* mentioned that their timing analysis may be realized using power/EM side-channel, because meaningfully similar two ciphertexts have meaningfully similar Hamming weights, resulting in similar power consumption/EM emanation. However, it is unknown how to exploit it with a sufficient accuracy in a practical setting/implementation (*e.g.*, the constant-time conditional move `cmov` on a microcontroller).[1]

2.2.2 Power/EM analysis

In [RRCB20], Ravi *et al.* showed an SCA on lattice-based KEMs. The attack is a side-channel-assisted CCA, in which an invalid ciphertext are generated such that the decrypted (or decoded) plaintext should be either 0 or 1 depending on a partial key. Here, the attacker cannot directly observe the plaintext due to FO transformation. However, the side-channel leakage during re-encryption is exploited to distinguish which the plaintext is 0 or 1, which allows the attack to estimate the partial key. Since the PRF fully randomizes the plaintext, the side-channel information during re-encryption considerably varies depending on which the plaintext is 0 or 1, that results in an exploitable side-channel leakage. The attacker can recover the full key of some lattice-based KEMs by querying the invalid ciphertext rotated to obtain information on different partial keys. Ravi *et al.* showed in [RRCB20] that their methodology is applicable to six lattice-based KEMs, namely, Kyber, Saber, FrodoKEM, Round5, NewHope, and LAC. Ravi *et al.* also presented a side-channel distinguisher based on a combination of t-test and reduced template, which yields a sufficiently feasible full-key recovery of the above six KEMs. Their distinguisher does not require the detailed knowledge of target implementation. In summary, this attack queries invalid ciphertexts to force the target device to execute a leaky plaintext depending on the underlying PKE. Although the plaintext is not directly available to the attacker due to FO transformation, side-channel leakage during re-encryption is exploited in order to estimate the plaintext and secret key.

Recently, in [BDH+21], Bhasin *et al.* reported SCA vulnerabilities of masked polynomial comparison schemes [OSPG18, BPO+20] for ciphertext equality check in lattice-based KEMs, and its application to Kyber. One of their attacks is based on the timing attack by Guo *et al.* [GTN20], and focuses on the leakage of masked polynomial comparison of $c = c'$ to realize a plaintext-checking oracle using a distinguisher made of t-test like the test vector leakage assessment (TVLA) [SLP05]. Note that, although the attack utilizes a plaintext-checking oracle as well as the proposed SCA in this paper, the literature mainly studies the (in)security of masked polynomial comparison for lattice-based KEMs, and they discussed only lattice-based KEMs and mentioned only Kyber, Saber, and FrodoKEM. In the sense, the contributions and goal of this paper are different from those of [BDH+21], as this paper mainly studies the generality and practicality of adaptive attacks using plaintext-checking oracle in the scenario of SCA on KEMs and presents a DL-based side-channel distinguisher generally applicable to various PRF implementations.

[1] Note that their attack and its power/EM variant are different from our proposed power/EM analysis as their attack focused on the Hamming distance between ciphertext and re-encryption result at the if statement in decapsulation algorithm in order to estimate how much similar they are. There has been no known practical power/EM analysis for realizing the plaintext-checking oracle because they did not show how to exploit the power/EM leakage of `cmov` in a practical setting nor any experimental validation. Recently, in [BDH+21], Bhasin *et al.* showed a power/EM SCA on masked polynomial comparison to realize the plaintext-checking oracle. Still, the attack is limited to some lattice-based KEMs and its generality is not discussed.

In addition, extended CCA SCA approach to lattice-based KEMs has been presented in [XPRO20, RBRC20, SKL+20, REB+21], and, in [NDGJ21], Ngo *et al.* presented an extended attack to a masked Saber implementation in [vBDK+21] using a DL technique. These attacks are very efficient in terms of the number of oracle accesses (*i.e.*, side-channel trace measurements) by employing chosen ciphertexts which result in more side-channel-leaky plaintext, regarding features of the underlying PKE and implementation. In other words, these attacks are very specific to the underlying PKE and its implementation. Although these attacks are CCA, they focus on some specific parts of the underlying PKE (*e.g.*, message encoding/decoding and number theoretic transform (NTT)-based multiplication) rather than FO transformation. In other words, these attacks achieve the efficiency focusing on a scheme/implementation-specific aspect; thus, they are less general in terms of KEM based on FO transformation.

Also, for code- and isogeny-based KEMs, there are some conventional SCAs (*e.g.*, [SKC+19, LNPS20] for code-based KEMs and [KAJ17, ZYD+20] for isogeny-based KEMs). However, these attacks focus on scheme/implementation-specific aspects rather than FO transformation, which indicates that these attacks are not attack on FO transformation and can be prevented using a PKE decryption (or DH) implementation with a countermeasure such as masking (as well as the above attacks on lattice-based KEMs).

As another attack direction, in [KPP20], Kannwischer *et al.* presented a single-trace SCA on SHAKE, which recovers the secret input to SHAKE by means of brief propagation (BP)-based method, so-called soft-analytical SCA (SASCA). Although their attack is powerful, its feasibility heavily depends on the word length of processor, key length (*i.e.*, the input bits to be recovered), and the signal-to-noise ratio (SNR) at the side-channel measurement. In fact, it is difficult to apply the attack to some practical settings (*e.g.*, 32-bit processor and longer-than 256-bit secret) regarding the post-quantum KEMs. In addition, the attack requires the detail of implementation to mount SASCA, and it can be prevented using a common SCA countermeasure (*e.g.*, masking). Note that Kannwischer *et al.* only showed the SCA on SHAKE, but not showed how to break KEMs instantiated with SHAKE.

3 Proposed Methodology

3.1 Plaintext-checking oracle

We first introduce a *plaintext-checking oracle*, which plays an essential role in the proposed attack. A plaintext-checking oracle is one of major oracles employed in adaptive attacks on a wide range of PKEs including lattice-, code-, and isogeny-based ones (*e.g.*, [GPST16, GTN20]). The key recovery attack using a plaintext-checking oracle is referred to as key recovery plaintext-checking attack (KR-PCA).

For a given KEM, let c be a valid ciphertext named *reference* ciphertext, and let m be the corresponding plaintext named *reference* plaintext. Note here that m denotes the PKE decryption result, rather than the output of KEM.Decaps. The attacker can obtain the reference ciphertext for any reference plaintext by performing the encapsulation. An adaptive attacker generates an invalid ciphertext c' which is a modification of c for an adaptive attack, and then queries it to the decryption oracle. Let m' be the plaintext corresponding to c'. An adaptive attack exploits the fact that there are two cases depending on the secret key: m' may be equal to the reference ciphertext m or other ciphertext $\hat{m}$. Formally, a plaintext-checking oracle $\mathcal{O}(c', m)$ returns 1 if $m = m'$; otherwise, it returns 0. For a KEM implementation based on FO transformation, such an oracle should be unavailable to any attacker, because the plaintext-checking oracle obviously leaks information on the PKE decryption result, which violates IND–CCA security guaranteed by FO transformation.

3.2 Proposed SCA

The proposed attack realizes a *plaintext-checking oracle* through a side-channel leakage to mount a chosen-ciphertext attack on the underlying CPA-secure PKE. In the proposed SCA, the attacker first observes the side-channel leakage during PRF execution of PKE.Decaps for the reference ciphertext c. Then, the attacker queries a modified ciphertext c' for an adaptive attack using plaintext-checking oracle, and observes the side-channel leakage during PRF execution in the re-encryption of decapsulation. If c' is decrypted to the reference plaintext m, the side-channel leakage for c' should be considerably similar to that for c because the PRF input is identical. In contrast, if c' is decrypted to other plaintext $\hat{m}$, two side-channel leakages should be meaningfully different. Thus, the attacker can distinguish whether the plaintext is a reference plaintext or other from the side-channel leakage of PRF. Since the proposed attack focuses on the PRF leakage, the proposed SCA can perform a key recovery even if the underlying PKE implementation has no secrecy leakage.

The proposed SCA consists of a profiling phase and attack phase. In the profiling phase, the attacker trains a classification model that distinguishes which the PRF input is the reference plaintext or other random plaintext from a side-channel trace in order to realize the plaintext-checking oracle as mentioned above. The trained model is called a side-channel distinguisher in this paper. In the attack phase, the attacker performs an adaptive attack on the CPA-secure PKE using the distinguisher as the plaintext-checking oracle. Note here that, although the attack employs a profiling phase, the proposed attack requires no profiling device because the profiling can be done using the target device without knowing the secret key, as well as the previous SCAs on lattice-based KEMs [RRCB20, XPRO20, RBRC20, SKL+20, NDGJ21]. In addition, the proposed SCA also does not require the detail of target implementation, because the plaintext-checking oracle can be realized by just comparing two traces, and we can perform the profiling using a DL technique without any assumption nor specific knowledge about the target implementation as described in Section 5. Such a side-channel distinguisher based on DL technique would be very suitable to two classification of traces for fixed vs. random input, as Moos *et al.* showed an efficient DL-based leakage assessment [MWM19].

4 Application to Post-Quantum KEMs

4.1 Lattice-based KEMs

4.1.1 Attack concept

To describe the underlying idea on adaptive attack on some of major lattice-based PKEs using a plaintext-checking oracle, we consider a lattice-based PKE with a simplified notation. Suppose that, in the PKE decryption, the plaintext before decoding is given in a form of $\mathsf{Encode}(m) + ke + e'$, where k is the secret key and e and e' are an error. Encode is an encode algorithm with a corresponding decode algorithm Decode to remove the noise $ke + e'$. Let c be a valid ciphertext corresponding to $\mathsf{Encode}(m) + ke + e'$, which can be computed by the encapsulation. For a lattice-based PKE, the ciphertext is correctly decrypted and decoded to m if the noise $ke + e'$ is less than a threshold value γ; otherwise, c is decrypted and decoded to other plaintext. The security of lattice-based PKE relies on the secrecy of the secret key, and PKE is usually designed such that the decryption failure probability is negligibly small.

In an adaptive attack, the attacker queries a modified ciphertext $c' = c + \delta$ to the decryption oracle, where δ is an error added to ciphertext. The modified cipertext decrypted to $\mathsf{Encode}(m) + ke + e' + \delta$ before decoding, in which $ke + e' + \delta$ is the noise to be removed. If $ke + e' + \delta < \gamma$, c' is decrypted and correctly decoded to m (*i.e.*, $m' = m$); otherwise (*i.e.*,

$ke + e' + \delta \geq \gamma$), c' is decrypted and wrongly decoded to other plaintext $\hat{m}$ (*i.e.*, $m' = \hat{m}$). In other words, if $ke + e' + \delta < \gamma$, $\mathcal{O}(c', m) = 1$; otherwise, $\mathcal{O}(c', m) = 0$. Therefore, the attacker knows $ke + e' + \delta$ by finding the value of δ such that $ke + e' + \delta = \gamma$ through adaptive queries to the plaintext-checking oracle. Thus, the attacker recovers the secret key k because e, e', δ, and γ are available to the attacker. Moreover, we can reduce the number of oracle accesses for a full-key recovery by querying a dedicated ciphertext, as mentioned in [BDL+19] (and described in the following sections).

4.1.2 FrodoKEM

We herein explicitly describe the adaptive attack on FrodoKEM in [GTN20] as a representative case. For the simplicity, we omit the detailed attack descriptions for Kyber and Saber because they are broken in a similar manner to FrodoKEM as described in Section 4.1.3. Although some instances of NTRU Prime and NTRU are also broken in a similar manner (we also omit the detail of these descriptions), we describe the implications in attacking them in Section 4.1.4 and Section 4.1.5, respectively.

Let $\mathbf{S}$ be the matrix for secret key, respectively. Let $\mathbf{S}'$, $\mathbf{E}$, $\mathbf{E}'$, and $\mathbf{E}''$ be the error matrices. When the ciphertext (c_0, c_1) corresponding to a pair of ciphertext matrices $\mathbf{B}'$ and $\mathbf{C}$ is input to the decryption oracle, the oracle computes the plaintext matrix $\mathbf{M}$ as

$$\begin{aligned}\mathbf{M} &= \mathbf{C} - \mathbf{B}'\mathbf{S} \\ &= \mathsf{Frodo.Encode}(m) + \mathbf{E}\mathbf{S}' - \mathbf{E}'\mathbf{S} + \mathbf{E}'',\end{aligned}$$

where $\mathsf{Frode.Encode}(m)$ denotes the encoded plaintext (or initial seed). The corresponding $\mathsf{Frode.Decode}(\mathbf{M})$ obtains m by removing the noise $\mathbf{E}\mathbf{S}' - \mathbf{E}'\mathbf{S} + \mathbf{E}''$ (which corresponds to $ke + e'$ in Section 4.1.1).

In the adaptive attack, the attacker generates a modified ciphertext consisting of c_0 and c_1' corresponding to $\mathbf{C} + \Delta$, where Δ is an error matrix added by the attacker (which corresponds to δ in Section 4.1.1). When querying (c_0, c_1'), the decryption oracle computes

$$\mathbf{M}' = \mathsf{Frodo.Encode}(m) + \mathbf{E}\mathbf{S}' - \mathbf{E}'\mathbf{S} + \mathbf{E}'' + \Delta.$$

Let the noise part $\mathbf{E}\mathbf{S}' - \mathbf{E}'\mathbf{S} + \mathbf{E}'' + \Delta$ denote $\mathbf{Q}$. Here, if all elements of $\mathbf{Q}$ are less than a threshold γ, $\mathbf{M}'$ is correctly decoded to m; otherwise, $\mathbf{M}'$ is wrongly decoded to other plaintext $\hat{m}$. Therefore, the attacker can find Δ such that $\Gamma = Q$ by adaptively querying (c_0, c_1') to the plaintext-checking oracle, where Γ is a matrix, all elements of which are a constant coefficient of γ. Because all elements in $\mathbf{Q}$ except for the secret key $\mathbf{S}$ (*i.e.*, $\mathbf{S}'$, $\mathbf{E}$, $\mathbf{E}'$, $\mathbf{E}''$, and Δ) are now available, the attacker recovers $\mathbf{S}$ by solving the linear equation $\Gamma = \mathbf{Q}$ if the attacker obtains Δ. Thus, Δ can be estimated using the plaintext-checking oracle, and the attacker can recover the secret key $\mathbf{S}$.

Algorithm 2 describes the attack on FrodoKEM using the plaintext-checking oracle. The attacker is supposed to determine a reference plaintext and the corresponding valid reference ciphertext by performing the encapsulation in advance. At Line 2, we initialize an $n \times \bar{n}$ matrix Δ as a zero matrix, where n and $\bar{n}$ denote the matrix size in FrodoKEM. We iteratively determine the (i, j)-th element of Δ (denoted by $\Delta_{i,j}$) at the loop of Lines 3–11. At Line 6, we queries modified ciphertexts $(c_0, c_1^{(i,j,\delta)})$, where $c_1^{(i,j,\delta)}$ is a ciphertext corresponding to a matrix of $\mathbf{C}$ where δ is added to the (i, j)-th element. If the (i, j)-th element of $\mathbf{Q}$ is less than γ, the corresponding plaintext matrix $\mathbf{M}'$ is correctly decoded to m (*i.e.*, $\mathcal{O}((c_0, c_1^{(i,j,\delta)}), m) = 1$); otherwise, $\mathbf{M}'$ is wrongly decoded to other plaintext (*i.e.*, $\mathcal{O}((c_0, c_1^{(i,j,\delta)}), m) = 0$). In particular, the (i, j)-th element of $\mathbf{Q}$ is equal to γ if and only if $\mathcal{O}((c_0, c_1^{(i,j,\delta)}), m) = 0$ and $\mathcal{O}((c_0, c_1^{(i,j,\delta-1)}), m) = 1$; thus, the attacker obtains information on $\Delta_{i,j}$ through the plaintext-checking oracle. After the attacker obtains $\Delta_{i,j}$ for all i and j, the attacker recovers the secret matrix $\mathbf{S}$ by solving the linear equation $\Gamma = \mathbf{Q}$ at Line 12.

Algorithm 2 Plaintext-checking key-recovery attack on FrodoKEM

Input: Reference ciphertext (c_0, c_1), reference ciphertext m, and noise matrices $\mathbf{S}'$, $\mathbf{E}$, $\mathbf{E}'$, and $\mathbf{E}''$
Output: Secret key sk (*i.e.*, Secret matrix $\mathbf{S}$)

1: **Function** AttackOnFrodoKEM$((c_0, c_1), m, \mathbf{S}', \mathbf{E}, \mathbf{E}', \mathbf{E}'')$
2: $\quad\Delta \leftarrow$ ZeroMatrix$(n, \bar{n})$;
3: $\quad$**for** $i = 0$ to $n - 1$ **do**
4: $\quad\quad$**for** $j = 0$ to $\bar{n} - 1$ **do**
5: $\quad\quad\quad$**for** $\delta \in \{0, 1, \ldots, \gamma - 1\}$ **do**
6: $\quad\quad\quad\quad$**if** $\mathcal{O}((c_0, c_1^{(i,j,\delta)}), m) = 0$ and $\mathcal{O}((c_0, c_1^{(i,j,\delta-1)}), m) = 1$ **then**;
7: $\quad\quad\quad\quad\quad\Delta_{i,j} \leftarrow \delta$;
8: $\quad\quad\quad\quad$**end if**
9: $\quad\quad\quad$**end for**
10: $\quad\quad$**end for**
11: $\quad$**end for**
12: $\quad$Solve linear equation $\Gamma = \mathbf{E}\mathbf{S}' - \mathbf{E}'\mathbf{S} + \mathbf{E}'' + \Delta$ about $\mathbf{S}$;
13: $\quad$**return** $\mathbf{S}$;
14: **end Function**

In Lines 5–9 requires, we require at most γ oracle accesses to determine $\Delta_{i,j}$ for each i and j if using a naive manner. However, as Guo *et al.* mentioned in [GTN20], we can reduce the number of oracle accesses to $\log \gamma$ by means of a binary search. Thus, Algorithm 2 achieves a full-key recovery with $n\bar{n} \log \gamma$ oracle accesses. In addition, we can further reduce the number of oracle accesses using a sparse ciphertext matrix as mentioned in [BDL+19]. Let us consider $\mathbf{D}^{(i)} = [\vec{0}, \ldots, \vec{0}, \vec{1}, \vec{0}, \ldots, \vec{0}]$, whose i-th column is all 1's. Suppose that we query a couple of ciphertext matrices $(\mathbf{D}^{(i)}, \mathbf{C})$. In the decryption, we have $\mathbf{M} = \mathbf{C} - \mathbf{D}^{(i)}\mathbf{S} = \mathbf{C} - Z$, where Z's first row is the i-th row of $\mathbf{S}$ and the rests are 0. We modify $\mathbf{C}$ and checks whether the decoded message is 0 or not as the plaintext-checking oracle. For example, let us consider the query $\mathbf{D}^{(1)}$ with $\mathbf{C}$ whose first row is filled by $q/2^{B+1}$ (where q is the modulus of the ring and B is the bit length of Frodo.Encode) and rest are filled by 0. We have $\mathbf{M}$ whose first row is $q/2^{B+1} - \mathbf{S}_{0,i}$ for $i = 0, \ldots, \bar{n} - 1$, which is decoded into 0 if and only if $\mathbf{S}_{0,i} > 0$. Thus, the attacker can directly recover the coefficients of secret matrix $\mathbf{S}$ with a less oracle accesses than the above straightforward attack.

In FrodoKEM.Decaps, the plaintext m' is first computed by PKE.Dec, and then SHAKE is computed for a concatenation of m' and a hash value associated with public key (denoted by **pkh** in the FrodoKEM document [A+20]). Since the SHAKE input is only dependant on m' and public key, the SHAKE execution in FrodoKEM.Decaps after the PKE decryption is exploitable *via* the proposed SCA.

4.1.3 Kyber and Saber

There are similar key-recovery attacks using sparse ciphertexts and plaintext-checking oracle against Kyber and Saber as that against FrodoKEM. For Kyber, the proposed SCA can recover the secret key on the basis of key-recovery attack against Kyber-512 in Round 2 by Huguenin-Dumittan and Vaudenay [HV20] (correctly speaking, we use the extended version in Xagawa *et al.* [XIU+21]). For Saber, the proposed SCA recovers the secret key on the basis of the adaptive attack in Huguenin-Dumittan and Vaudenay [HV20] for LightSaber and the attack by Osumi *et al.* [OUKT21] for Saber and FireSaber. In all cases, the decrypted plaintext is 0^ℓ or a unit vector $0^{i-1} \parallel 1 \parallel 0^{\ell-i-1}$. The plaintext-checking oracle can be implemented using the PRF leakage in the re-encryption as well as FrodoKEM.

4.1.4 NTRU Prime

NTRU Prime has two KEM schemes sntrupr (Streamlined NTRU Prime) and ntrulpr (NTRU LPRime). NTRU LPRime has a similar structure to Kyber, Saber, and FrodoKEM. Thus, we

can mount a key-recovery plaintext-checking attack as in [XIU+21]. In their attack, the decrypted plaintext is 1^ℓ or a vector of the form $1^{i-1} \parallel 0 \parallel 1^{\ell-i-1}$ for i.

Streamlined NTRU Prime has a similar structure to NTRU. A plaintext is r and a ciphertext is $c = \mathsf{Round}(h \cdot r)$, where $\mathsf{Round}(x)$ rounds each coefficient of x to a nearest element in $3\mathbb{Z}$.[2] However, there are some technical hurdles to adapt key-recovery plaintext-checking attacks against NTRU. For example, Streamlined NTRU Prime's PKE.Dec internally checks the Hamming weight of a decrypted plaintext r and overwrite the decrypted plaintext with the fixed plaintext r_{fixed} if the test failed. Very recently, Ravi *et al.* [REB+21] proposed two key-recovery side-channel attacks against Streamlined NTRU Prime, which is inspired by chosen-ciphertext attacks against NTRU by Jaulmes and Joux [JJ00].

The one is based on the 'plaintext-checking' oracle, which tests if the internal variable is 0 or not. The internal decrypted plaintext is 0 or some polynomial and the Hamming weight of them are not correct. Hence, the output of the underlying PKE.Dec is r_{fixed} in both cases. Thus, in order to implement this 'plaintext-checking' oracle, we need to analyze side-channel information of the computation in $\mathsf{PKE.Dec}(\mathsf{sk}, c)$, which is out of focus of this paper.

The other is based on the decryption-failure oracle, which tests if the decrypted plaintext is intended r or not. In the case, the Hamming weight of r is proper. On the other hand, if decryption failure occurs, the Hamming weight of decrypted plaintext becomes invalid and it is overwritten by r_{fixed}. Ravi *et al.* implemented the decryption-failure oracle by analyzing side-channel information of the re-encryption test. We adopt their attack in our context.

Ravi *et al.*'s DF-based attack and our modification: Their attack proceeds two phases:

1. In the first phase, we seek δ which occurs 'collision' of the secret key by checking the decryption of $c' = c + \delta$, where $c = \mathsf{Round}(h \cdot r_{\mathsf{valid}})$ for a correct plaintext r_{valid}, is r_{valid} or r_{fixed}. If the decryption failure is detected, then we employ δ as c_{base}. They design the structure of δ carefully. We slightly change the structure of δ to boost the success probability to get '1-collision'.[3] We then follow their strategy to design δ and estimate the success probability to get appropriate δ to approximately 1% and 1.5% for sntrup653 and sntrup1277, respectively.[4] See [REB+21, Section 4.1 and 4.2] for the details.

2. In the second phase, we query four ciphertexts modifying c and c_{base} and check the decrypted results are r_{valid} or r_{fixed} to determine a coefficient of the secret key.

We note that NTRU Prime uses a variant of the FO transformation, which does not involve the computation of randomness because the underlying PKE.Enc is *deterministic* as NTRU. Fortunately, NTRU Prime uses the explicit re-encryption test. Moreover, NTRU Prime adds an additional hash $\mathsf{HashConfirm}(r, \mathsf{pk})$ to its ciphertext of the underlying PKE, where $\mathsf{HashConfirm}(r, \mathsf{pk}) = \mathsf{Hash}(\mathsf{0x02} \parallel \mathsf{Hash}(\mathsf{0x03} \parallel r) \parallel \mathsf{Hash}(\mathsf{0x04} \parallel \mathsf{pk}))$ and $\mathsf{Hash}(z)$ is the first 32 bytes of $\mathsf{SHA\text{-}512}(z)$. Thus, the decapsulation algorithm computes $\mathsf{HashConfirm}(r', \mathsf{pk})$ in the re-encryption test, which leaks side-channel information of r' as we wanted.

[2] Letting $m = c - h \cdot r$, we can write $c = h \cdot r + m$ as in the NTRU case.

[3] We recommend the parameter setting $(m, n) = (1, 3)$ for sntrup653 and sntrup1277 as Ravi *et al.* , while we change the range of $i_1, \ldots, i_m, j_1, \ldots, j_n \in [\lfloor p/2 \rfloor, p)$ instead of $[0, p)$. This simple trick approximately *triples* a probability of 1-collision.

[4] We recommend the parameter setting $(k_1, k_2) = (96, 282)$ and $(152, 486)$ for sntrup653 and sntrup1277, respectively. Approximately 20% of noise $n'[i]$ makes $a[i] > q/2$.

4.1.5 NTRU

NTRU has two KEM schemes NTRU-HPS and NTRU-HRSS, which are slightly different. In PKE of both KEMs, a public key is h, a plaintext is a pair of 'short' polynomials (r, m), and a ciphertext is $c = h \cdot r + \mathsf{Lift}(m) \in \mathbb{Z}[x]/(q, x^n - 1)$, where Lift is a bijection. NTRU's ciphertext space is $\{\, c \in \mathbb{Z}[x]/(q, x^n - 1) \mid c \equiv 0 \pmod{(q, x-1)} \,\}$, which is isomorphic to $\mathbb{Z}[x]/(q, (x^n - 1)/(x - 1))$.

We can modify the key-recovery plaintext-checking attack by Hoffstein and Silverman [HS99] and Jaulmes and Joux [JJ00] against the original NTRU, in which m is a plaintext and r is a randomness. Those key-recovery attacks modify $c = h \cdot r + \mathsf{lift}(m)$ into $c' = c + \delta$ and check if a *half* m of decrypted plaintext (r, m) is equivalent to the expected half plaintext m_{guess}, say, 0, or not. We note that the rest half r_{guess} of the expected plaintext can be computed from the ciphertext and the expected half plaintext by $r_{\mathsf{guess}} = (c' - \mathsf{lift}(m_{\mathsf{guess}})) \cdot h^{-1}$. The main hurdle to adapt this attack into NTRU is that NTRU's ciphertext space is changed from the original NTRU. Hence, δ also should satisfy $\delta \equiv 0 \pmod{(q, x-1)}$. This constraint makes analysis complex and we do not select those attacks.

We can also use the key-recovery plaintext-checking attack against NTRU-HPS by [DDS+19] and that against NTRU-HRSS by [ZCQD21]. Those key-recovery attacks fix $m_{\mathsf{guess}} = 0$, modify r' and r_{guess}, compute $c = h \cdot r'$, and check if the decrypted plaintext (r, m) is equivalent to the guess $(r_{\mathsf{guess}}, m_{\mathsf{guess}})$ or not. In those attacks, c satisfies $c \equiv 0 \pmod{(q, x-1)}$ because $h \equiv 0 \pmod{(q, x-1)}$ by design.

We note that NTRU in Round 2 and 3 uses SXY [SXY18] as the variant of the FO transformation, which does not involve the computation of $r' \leftarrow G(m')$ because the underlying PKE.Enc is *deterministic*. Moreover, NTRU does not perform the re-encryption test explicitly. (Un)fortunately, NTRU's decapsulation program in pqm4 computes both keys $k = \mathsf{KDF}(r, m)$ and $k' = \mathsf{KDF}(s, c)$ and outputs one of them according to the result of the implicit re-encryption test. Fortunately, in our experiment, we can detect $m_{\mathsf{guess}} = 0$ or not from those computations with high accuracy. (See Section 6 for the details.)

4.2 Code-based KEMs

4.2.1 HQC

Roughly speaking, HQC has a similar structure to lattice-based KEM schemes Kyber, Saber, FrodoKEM, and NTRU LPRime, while HQC is based on the code problem. Hence, we can adapt key-recovery plaintext-checking attacks against them into that against HQC. Indeed, Huguenin-Dumittan and Vaudenay [HV20] gave a key-recovery plaintext-checking attack against HQC in Round 2 by mimicking that by Băetu *et al.* [BDL+19] against another code-based PKE Lepton [YZ17]. Although HQC changed parameters and decoder from Round 2 to Round 3, we can still perform the key-recovery plaintext-checking attack by adjusting the parameter setting. See Xagawa *et al.* [XIU+21] for the detail. In their attack, the decrypted plaintext is 0^ℓ or a vector of the form $0^{i-1} \parallel 1 \parallel 0^{\ell-i-1}$ for some i. As HQC employs SHAKE for the decrypted plaintext in the re-encryption, the plaintext-checking oracle can be realized using the PRF leakage through the proposed framework.

4.2.2 BIKE

BIKE in Round 3 has a single KEM scheme based on the Niederreiter PKE with quasi-cyclic moderate density parity-check (QC-MDPC) code. Guo *et al.* [GJS16] gave a key-recovery reaction attack against QC-MDPC [MTSB13], which is a variant of the McEliece PKE with QC-MDPC codes. Roughly speaking, by using the decryption oracle, one can recover distance profile $\mu(h_0)$ of a half of a secret key $h_0 \in \mathrm{GF}(2)^n$, where distance profile contains (d, μ_d) for $d = 1, 2, \ldots, n/2$ that implies there are μ_d pairs of 1's with distance d in h_0.

Guo *et al.* reported that one can recover h_0 from its distance profile $\mu(h_0)$ in the parameter set for 80-bit security in practice. Xagawa *et al.* [XIU+21] reported the GJS attack [GJS16] against QC-MDPC can be *partially* applied to BIKE in round 3 in the presence of the plaintext-checking oracle. They recover approximately quarter of the distance profile in the parameter set for 128-bit security. The decapsulation of BIKE employ the PRF in the re-encryption (*i.e.*, AES and SHA384), which is exploited to implement the plaintext-checking oracle *via* the proposed framework.

Note that the GJS attack queries many ciphertexts from crafted invalid plaintexts *at random* in order to compute (d, μ_d), say, 2000 ciphertexts for each d, and checks if they are decrypted correctly or not. Hence, we cannot fix the template plaintext.

4.2.3 Classic McEliece

The proposed attack is not applicable to Classic McEliece because no adaptive attack on the PKE of Classic McEliece is known. However, regarding the decapsulation of Classic McEliece, we can realize a plaintext-checking oracle for the PKE because Classic McEliece computes an additional hash $\mathsf{Hash}(2, m')$ in the re-encryption test as Streamlined NTRU Prime, which indicates that the proposed attack can be mounted on Classic McEliece if a key-recovery plaintext-checking attack is discovered.

4.3 Isogeny-based KEM

Hereafter, we introduce the SCA on SIKE based on the active attack on Jao's and De Fao's supersingular isogeny cryptosystem [JDF11] (namely, supersingular isogeny Diffie–Hellman (SIDH)) presented by Galbraith *et al.* [GPST16], with a consideration and modification for our SCA to mount it on SIKE.Decaps.

Let P_A, Q_A, P_B, and Q_B be the public generator points on E_0, where E_0 is the starting Montgomery curve $y = x^3 + 6x^2 + x$ over $\mathbb{F}_{p^2}$ with $p = 2^{e_A}3^{e_B} \pm 1$. Let sk_2 and sk_3 be Alice's and Bob's secret key, respectively. Let $R_A = P_A + [\mathsf{sk}_2]Q_A$ and $R_B = P_B + [\mathsf{sk}_3]Q_B$ be the secret point to generate a finite cyclic group for the kernel of Alice's and Bob's isogeny ϕ_A and ϕ_B, respectively. As well, let pk_2 and pk_3 be the public keys. At SIKE.Encaps, the attacker first generates a reference ciphertext (c_0, c_1), where the reference j-variant for the ciphertext corresponds to a curve $E_0/\langle R_A, R_B\rangle$. Here, $E_0/\langle R_A, R_B\rangle$ denotes the shared curve isogenous to E_0 with regard to an isogeny with a kernel of a finite group $\langle R_A, R_B\rangle \coloneqq \{\, [n_A]R_A + [n_B]R_B \mid n_A \in \{1, 2, \ldots, 2^{e_A} - 1\}, n_B \in \{1, 2, \ldots, 3^{e_B} - 1\}\,\}$. Since sk_3 acts as the secret key in SIKE, the goal of an active attacker is to recover sk_3 by adaptively querying ciphertexts to the decryption oracle.

Let us consider the ternary digit representation of the secret key as $\mathsf{sk}_3 = 3^0\beta_0 + 3^1\beta_1 + \cdots + 3^i\beta_i + \cdots + 3^{e_B-1}\beta_{e_B-1}$ where $\beta_i \in \{0, 1, 2\}$. Let $E_A = E_0/\langle R_A\rangle$ be Alice's public curve isogenous to E_0 with regard to Alice's isogeny ϕ_A with a kernel $\langle R_A\rangle$, and let $\tilde{P_A} = \phi_A(P_B)$ and $\tilde{Q_A} = \phi_A(Q_B)$ denote Bob's public points on E_A (calculated by Alice). The valid ciphertext of SIKE.Encaps is given as $c_0 = \mathsf{pk}_2 = (E_0/\langle R_A\rangle, \tilde{P_A}, \tilde{Q_A})$ [5] and $c_1 = m \oplus \mathsf{SHAKE}(j(E_0/\langle R_A, R_B\rangle))$, where m is a random number from $U(\{0,1\}^n)$ with $n \in \{128, 192, 256\}$. In the adaptive attack, the attack first generates three invalid ciphertexts $(c_0^{(\tau)}, c_1)$ for $\tau \in \{0, 1, 2\}$, where $\tilde{P_A}$ and $\tilde{Q_A}$ in c_0 are replaced with

$$\tilde{P}_A^{(\tau)} = \tilde{P_A} - [3^{e_B-1}\tau]\tilde{Q_A},$$
$$\tilde{Q}_A^{(\tau)} = \tilde{Q_A} + [3^{e_B-1}]\tilde{Q_A},$$

respectively (Note that $\tilde{Q}_A^{(\tau)}$ is not dependant on τ).

[5] In practice, instead of $E_0/\langle R_A\rangle$, the ciphertext (or public key) of SIKE is given by three points $(\tilde{D_A}, \tilde{P_A}, \tilde{Q_A})$, where $\tilde{D_A} = \tilde{P_A} - \tilde{Q_A} = \phi_A(P_B - Q_B)$. $E_0/\langle R_A\rangle$ is reconstructed from the three points by the receiver. This has no influence on the adaptive attack.

Then, let $R_{AB} = \tilde{P_A} + [\mathsf{sk}_3]\tilde{Q_A}$ be the cyclic group generator of the isogeny kernel at SIKE.Decaps corresponding to the reference ciphertext (c_0, c_1). For the modified ciphertexts, the PKE decryption calculates $R_{AB}^{(\tau)}$ for $\tau \in \{0, 1, 2\}$ such that

$$\begin{aligned} R_{AB}^{(\tau)} &= (\tilde{P_A} - [3^{e_B-1}\tau]\tilde{Q_A}) + [\mathsf{sk}_3](\tilde{Q_A} + [3^{e_B-1}]\tilde{Q_A}) \\ &= R_{AB} + [3^{e_B-1}(\mathsf{sk}_3 - \tau)]Q_A, \end{aligned}$$

and then computes the j-variant of $E_A/\langle R_{AB}^{(\tau)}\rangle$. Here, it holds $R_{AB} = R_{AB}^{(\tau)}$ if and only if $\tau = \beta_0$, because

$$[3^{e_B-1}(\mathsf{sk}_3 - \tau)]\tilde{Q_A} = [3^{e_B-1}(\beta_0 - \tau)]\tilde{Q_A},$$

which follows from the fact that the order of $\tilde{Q_A}$ is 3^{e_B}. Therefore, if the attacker can know whether $R_{AB}^{(\tau)}$ leads to the j-variant value corresponding to $E_0/\langle R_A, R_B\rangle$ (*i.e.*, the same value as that of reference ciphertext (c_0, c_1)), she obtains the least significant ternary digit of the secret key (*i.e.*, β_0).

The full-key recovery is achieved by repeating the above attack in an iterative manner. Let us consider the recovery of the i-th ternary digit (*i.e.*, β_i), supposing that the attacker has already recovered up-to the $(i-1)$-th digit (*i.e.*, $\beta_0, \beta_1, \ldots, \beta_{i-1}$). Let $K_i = 3^0\beta_0 + 3^1\beta_1 + \cdots + 3^{i-1}\beta_{i-1}$ $(K_0 = 0)$ be the recovered part of the secret key. The attacker generates the modified ciphertexts $(c_0^{(\tau,i)}, c_1)$ for $\tau \in \{0, 1, 2\}$, where $\tilde{P_A}$ and $\tilde{Q_A}$ in c_0 are replaced with

$$\begin{aligned} \tilde{P}_A^{(\tau,i)} &= \tilde{P_A} - [3^{e_B-i-1}K_i + 3^{e_B-1}\tau]\tilde{Q_A}, \\ \tilde{Q}_A^{(\tau,i)} &= \tilde{Q_A} + [3^{e_B-i-1}]\tilde{Q_A}, \end{aligned}$$

respectively. When querying $(c_0^{(\tau,i)}, c_1)$ to the decryption oracle, the generator of cyclic group is calculated as

$$\begin{aligned} R_{AB}^{(\tau,i)} &= (\tilde{P_A} - [3^{e_B-i-1}K_i + 3^{e_B-1}\tau]\tilde{Q_A}) + [\mathsf{sk}_3](\tilde{Q_A} + [3^{e_B-i-1}]\tilde{Q_A}) \\ &= R_{AB} + [3^{e_B-i-1}(\mathsf{sk}_3 - K_i) - 3^{e_B-1}\tau]\tilde{Q_A} \\ &= R_{AB} + [3^{e_B-1}(\beta_i - \tau)]\tilde{Q_A}, \end{aligned}$$

which implies that $R_{AB}^{(\tau,i)}$ should equal to R_{AB} if and only if $\tau = \beta_i$. Thus, the attack recovers β_i through the decryption oracle that tells whether the j-variant of $(c_0^{(\tau,i)}, c_1)$ is equal to that of the reference ciphertext (c_0, c_1); and the full-key recovery is completed within the number of oracle accesses linear to e_B.

Algorithm 3 illustrates the SCA on SIKE. In SIKE.Decaps, the j-variant value depends on only c_0 (but not c_1), and the PKE decryption result is always identical for a fixed j-variant and c_0. Therefore, we can realize the plaintext-checking oracle for SIKE through the side-channel leakage form G (*i.e.*, SHAKE) because we should distinguish whether the input to G is reference plaintext m or other. Algorithm 3 uses the plaintext-checking oracle $\mathcal{O}$ (*i.e.*, side-channel distinguisher herein) at Line 8. Thus, the number of distinguisher call for the full-key recovery is at most $3e_B$. Note that it can be reduced to $2e_B$ because the attacker knows $\beta_i = 2$ without querying $(c_0^{(2,i)}, c_1)$ if $\mathcal{O}((c_0^{(0,i)}, c_1), m) = 0$ and $\mathcal{O}((c_0^{(1,i)}, c_1), m) = 0$. It is also possible to use the side-channel leakage of SHAKE inside the PKE decryption (denoted by F in the SIKE documentation [J+20]) instead of G at the decapsulation. Note also that this attack can be readily extended to SIKE over $\mathbb{F}_{p^2}$ with $p = \ell_A^{e_A}\ell_B^{e_B} f \pm 1$ by replacing the base of coefficients (*i.e.*, "3" in the above equations and Algorithm 3) with ℓ_B and examining τ from $\{0, 1, \ldots, \ell_B - 1\}$.

Although we describe an adaptive attack using a naïve plaintext-checking oracle, we can further reduce the number of oracle accesses by using a more sophisticated plaintext-checking oracle. Recall that, when querying $(c_0^{(\tau,i)}, c_1)$ to the decryption oracle, the

Algorithm 3 Plaintext-checking key-recovery attack on SIKE

Input: Reference ciphertext (c_0, c_1) and reference plaintext m
Output: Secret key sk_3

Function AttackOnSIKE($(c_0, c_1), m$)
  $K_0 \leftarrow 0$;
  for $i = 0$ to $e_B - 1$ **do**
    for each $\tau \in \{0, 1, 2\}$ **do**
      $\tilde{P}_A^{(\tau,i)} \leftarrow \tilde{P_A} - [3^{e_B-i-1}K_i + 3^{e_B-1}\tau]\tilde{Q_A}$;
      $\tilde{Q}_A^{(\tau,i)} \leftarrow \tilde{Q_A} + [3^{e_B-i-1}]\tilde{Q_A}$;
      $(c_0^{(\tau,i)}, c_1) \leftarrow ((E_A, \tilde{P}_A^{(\tau,i)}, \tilde{Q}_A^{(\tau,i)}), c_1)$;
      if $\mathcal{O}((c_0^{(\tau,i)}, c_1), m) = 1$ **then**
        $\beta_i \leftarrow \tau$;
        $K_{i+1} \leftarrow K_i + 3^i\beta_i$;
      end if
    end for
  end for
  return K_{e_B};
end Function

generator of cyclic group is calculated as

$$R_{AB}^{(\tau,i)} = R_{AB} + [3^{e_B-1}(\beta_i - \tau)]\tilde{Q_A}.$$

If we set $\tau = 0$, then we will get either R_{AB}, $R_{AB} + [3^{e_B-1}]\tilde{Q_A}$, or $R_{AB} + [2 \times 3^{e_B-1}]\tilde{Q_A}$, depending on β_i. Note that all the candidate values are independent of i. Therefore, we can estimate β_i from side-channel trace by distinguishing which the PRF input value corresponds to R_{AB}, $R_{AB} + [3^{e_B-1}]\tilde{Q_A}$, or $R_{AB} + [2 \times 3^{e_B-1}]\tilde{Q_A}$ when querying $(c_0^{(0,i)}, c_1)$. To do so, we employ a side-channel distinguisher that performs a 3-classification for R_{AB}, $R_{AB}+[3^{e_B-1}]\tilde{Q_A}$, or $R_{AB}+[2\times3^{e_B-1}]\tilde{Q_A}$, instead of 2-classification for $\mathcal{O}((c_0, c_1), (c_0^{(0,i)}, c_1))$. Since the attacker can compute R_{AB}, $R_{AB} + [2 \times 3^{e_B-1}]\tilde{Q_A}$, and $R_{AB} + [2 \times 3^{e_B-1}]\tilde{Q_A}$ without knowing secret key, the attacker can query the ciphertexts corresponding to them, acquire side-channel traces for profiling (*i.e.*, training dataset), and perform a profiling (*i.e.*, NN training) for the 3-classification to implement the oracle. Thus, we can determine β_i from one query with an ℓ_B-classification and no need to modify τ, and we can perform a full-key recovery of SIKE using e_B oracle accesses, independently of ℓ_B.

4.4 Complexity analysis

Table 2 reports the number of oracle accesses required for the proposed SCA to recovery the full key of NIST PQC third-round candidates for KEM. For the simplicity, Table 2 only shows the results for instances with a security level equivalent to AES128 and AES256 (*i.e.*, NIST security levels 1 and 5, respectively).

From Table 2, we confirm that the key recovery can be realized with a sufficiently feasible number of oracle accesses. Although BIKE level 1 requires 3M oracle accesses for a partial-key recovery as the hardest case, most KEMs can be broken within 60,000 oracle accesses. Here, Kyber, Saber, NTRU, NTRU Prime, and SIKE are less complex than the code-based KEMs. This may be because the number of secret coefficients to be recovered *via* plaintext-checking oracle is greater for the code-based KEMs. The proposed SCA would be still feasible, as the modern SCAs (on symmetric ciphers) are evaluate with (the order of) more-than 10M or 100M traces (*e.g.*, [SM15, SM19, SBM19]).

In comparison with previous CCA SCAs on lattice-based KEMs (*e.g.*, [XPRO20, RBRC20, SKL+20, NDGJ21]), the proposed attack requires more oracle accesses. This is because these previous SCAs exploit scheme/implementation-specific aspects for improved efficiency in terms of the number of traces. Moreover, the attack by Ravi *et al.* [RRCB20]

Table 2: Worst-case number of oracle accesses required for proposed attack to recover full key (except for BIKE and Classic McEliece)

KEM type	Scheme	Instance	# Oracle accesses
Lattice	Kyber	Kyber-512	1536 ($= 3 \times 512$)
		Kyber-1024	3072 ($= 3 \times 1024$)
	Saber	LightSaber-KEM	3072 ($= 4 \times 512 + 2 \times 512$)
		FireSaber-KEM	3072 ($= 3 \times 1024$)
	FrodoKEM	FrodoKEM-640	25600 ($= 5 \times 5120$)
		FrodoKEM-1344	43008 ($= 4 \times 10752$)
	NTRU Prime	ntrulpr653	1306 ($= 2 \times 653$)
		ntrulpr1277	2554 ($= 2 \times 1277$)
		sntrup653	2712 in avg. ($= 100/1 + 4 \times 653$)
		sntrup1277	5175 in avg. ($= 100/1.5 + 4 \times 1277$)
	NTRU	ntruhrss701	$\approx$ 2804 ($= 4 \times 701$)
		ntruhps2048509	$\approx$ 1018 ($= 2 \times 509$)
		ntruhps4096821	$\approx$ 1642 ($= 2 \times 821$)
Code	HQC	hqc128	≈ 18111 ($= 46 + \lg(46) + 46 \times (384 + \lg(384))$)
		hqc256	≈ 58536 ($= 90 + \lg(90) + 90 \times (640 + \lg(640))$)
	BIKE[†]	Level 1	3M ($= 2000 \times 1500$)
		Level 5	N/A
	Classic McEliece	Any	N/A
Isogeny	SIKE	SIKEp434	274 ($= 2 \times 137$)
		SIKEp751	478 ($= 2 \times 239$)

[†] Denote 1500 distance profiles out of 6162 full distance profiles for Level 1. It is difficult to reliably recovery key bits more than this through the proposed SCA.

can also perform a full-key recovery of Kyber, Saber, and FrodoKEM, although the SCA is applicable to (relatively) black-box implementation. We stress that the major advantage of the proposed attack is the generality, as the proposed attack is applicable to many lattice-, code-, and isogeny-based KEMs, even if its implementation is (relatively) black-box. Due to the high applicability of adaptive attack using the plaintext-checking oracle, the proposed attack offers a higher generality for KEMs based on FO transformation and its variants.

5 Side-Channel Distinguisher Design

This subsection describes the design of a DL-based side-channel distinguisher. Several studies have been evaluated and demonstrated the significant advantage of DL in SCA in recent years (*e.g.*, [BPS+18, KPH+19, PCP20, ZBHV20, WAGP20]). In many previous works, the DL technique is employed to perform an efficient profiling SCA. One of major advantages of DL-based profiling SCA (compared to conventional profiling SCA such as template attack) is that it does require the detail of target implementation nor any assumption of side-channel leakage. In the DL-based profiling SCA, a trained NN is used to estimate (Hamming weight/distance of) intermediate value from side-channel leakage, and the secret key is estimated using the likelihood from the output of NN (*i.e.*, probability for (Hamming weight/distance of) intermediate value). Therefore, the conventional DL-based SCA on AES implementation basically has 9 outputs at the output layer corresponding to Hamming weight/distance classification or 256 outputs for intermediate value classification.

The previous studies have developed many NN models to efficiently perform the key recovery with less traces [ZBHV20, WAGP20]. In the following experiment, we employ a convolutional NN (CNN), as several previous studies showed its practicality and effectiveness in DL-based SCA. Our CNN in the experiment is designed to have a sufficient model capacity such that it can be applied to various PRF implementation including software and hardware with and without masking countermeasure. The proposed SCA requires two-classification of whether the input to PRF is the reference plaintext or not,

and therefore we construct a CNN model, the output layer of which is with an activation function of Sigmoid and has one output. Note that there is a possibility that we can exploit a conventional NN presented in previous studies on DL-based SCA for the proposed attack by means of fine tuning or transfer learning. However, in this paper, we used a standard CNN model for the generality, as some of conventional NNs for DL-based SCA are specified for target implementations (*e.g.*, [WAGP20]).

For a successful key recovery, we require a very accurate model to realize a perfect plaintext-checking oracle, because an error in plaintext-checking oracle should render the recovered key critically incorrect. However, the accuracy of an NN model for SCA may be sometimes nonnegligibly low due to the presence of noise and SCA countermeasure. To improve the accuracy of plaintext-checking oracle realized by the model, a simple method is to use multiple traces for one plaintext-checking oracle. More concretely, the attacker acquires t traces for a modified ciphertext c', performs an inference for each trace, and then estimates the input to PRF as the majority vote of the inference results. Let a be the accuracy of the model. If using t traces for an oracle, the expected accuracy of the oracle realized by such a majority vote of multiple NN outputs, denoted by α_t, is given by

$$\alpha_t = 1 - \sum_{s=0}^{\lceil t/2 \rceil} \binom{t}{s} a^s (1-a)^{t-s}. \tag{1}$$

We can determine t such that the success rate of whole attack is larger than a threshold σ; that is, $\sigma \leq {\alpha_t}^u$, where u is the number of required oracle accesses shown in Table 2. This is because the attacker requires completely correct plaintext-checking oracles for a correct key recovery. We quantitatively evaluate the number of required side-channel traces for a convincing key recovery success rate in Section 6.

However, such a majority vote considers the NN output as a binary value, although the NN output is given as a probability of the input to PRF being the reference plaintext, which indicates that the majority vote does not fully exploit the advantage of the NN-based distinguisher. As an efficient alternative, we determine the plaintext-checking oracle output by means of the likelihood comparison, which indicates that the label is determined according to the negative log likelihood (NLL) for a hypothetical oracle output of $b \in \{0, 1\}$ as

$$\mathrm{NLL}_b(q, \hat{\theta}) = -\frac{1}{t} \sum_{s=1}^{t} \log q(b \mid \boldsymbol{x}_s; \hat{\theta}), \tag{2}$$

where q denotes a probability distribution parameterized by $\hat{\theta}$ (*i.e.*, the trained NN), and $\boldsymbol{x}_s$ is the s-th side-channel trace. Since such methods exploit the NN output compared to the above majority vote, these methods would achieve a higher accuracy in realizing the plaintext-checking oracle. In fact, if $q(b \mid \boldsymbol{x}_s; \hat{\theta})$ has been sufficiently trained and mimics the true distribution, such a likelihood ratio test (the NLL herein) becomes the most powerful test according to the Neyman–Pearson lemma [NP33]. One major drawback of this method is that it is quite difficult to evaluate the resulting accuracy in an analytical manner; therefore, we experimentally evaluate it to demonstrate its practicality and effectiveness through the evaluation of success rate of the whole attack on KEMs using actual NN models trained for the experimental implementations.

6 Experimental Validation

6.1 Experimental setup

In the following experiments, we employed CUDA 11.0, cuDNN 8.0.5, Tensorflow-gpu 2.4.1, and Keras 2.4.0 on an Intel Xeon W-2145 3.70 GHz and NVIDIA GeForce GTX 2080 for

Table 3: CNN hyper parameters

	Input	Operator	Output	Activation function	Batch normalization	Pooling	Stride
Conv1	1000×1	conv1d (3)	4	SELU	Yes	Avg (2)	2
Conv2	500×4	conv1d (3)	4	SELU	Yes	Avg (2)	2
Conv3	250×4	conv1d (3)	4	SELU	Yes	Avg (2)	2
Conv4	250×8	conv1d (3)	8	SELU	Yes	Avg (2)	2
Conv5	125×8	conv1d (3)	8	SELU	Yes	Avg (2)	2
FLT	64×8	flatten	496	-	-	-	-
FC1	496	dense	20	SELU	No	No	-
FC2	20	dense	20	SELU	No	No	-
FC3	20	dense	1	Sigmoid	No	No	-

the NN training. We employed the NLL as a loss function for training. The learning rate was 0.001, the batch size was 32, and the number of epochs was 100. Table 3 summarizes its hyper parameters of the CNN for traces with 1,000 sample points, where the top and bottom columns denote the input and output layer, respectively, and the remaining hidden layers are connected in the ascending order from the input to output.

Here, $S_1 \times S_2$ in the row of "Input" denotes the input shape as S_1 is the traces size and S_2 is the input dimension, and conv1d(F) in the row of "Operator" denotes the operation at the each layer as F is the filter size. The CNN consists five convolutional layers *Conv1*, *Conv2*, ..., and *Conv5* followed by three fully connected layers *FC1*, *FC2*, and *FC3*. The CNN has two outputs. Given a side-channel trace, *FC3* outputs the probability that plaintext (*i.e.*, PRF input) is equal to the reference plaintext (or other plaintext in contrary).

For acquiring the side-channel traces, we used an oscilloscope Keysight Technologies MSOX6004A to measure the power consumption or EM emanation. We employ the following four PRF implementations: non-protected AES and SHAKE software, masked AES software, and masked AES hardware. Although SHAKE is rather common as PRF in FO transformation than AES, we herein target masked AES software/hardware since many SCA countermeasures for symmetric key primitives have been developed with consideration and application to AES rather than SHAKE (As far as we know, there is no publicly available masked SHAKE implementation). However, there would be little difference in the attack result between AES and SHAKE, if they are protected using an identical masking scheme. The detail of each implementation is described below.[6]

Non-protected AES and SHAKE software. We implement an AES and SHAKE software provided in `pqm4` on an STM32F415RGT6 equipped with a NewAE Technology STM32F Target Board. These software employ no countermeasure against SCAs. The supply voltage current is observed as the side-channel trace by means of a NewAE Technology chip-whisperer CW308. We used 20,000 traces for training and used 10,000 traces for validation among 20,000 traces. We also used another 10,000 traces for test. For AES, the plaintext is given by a fixed or random value, and the key is fixed for both fixed and random plaintexts.[7] For SHAKE, the input is given as same as the plaintext.

[6] We also implemented and evaluated an NTRU software of `ntruhrss701` in `pqm4` in the setting same as non-protected AES/SHAKE software. We targeted the procedure corresponding to re-encryption in `owcpa_dec`, instead of a PRF in NTRU.Decaps as a KDF, whereas the key recovery of NTRU can be also performed using the leakage of KDF as described in Section 4.1.5. As a result, we confirmed that the DL-based distinguisher achieves a 99.6% accuracy, and its combination with likelihood comparison achieves a 100% test accuracy with only three traces.

[7] In using AES-256-CTR as an XOF for modern KEMs (*e.g.*, Kyber, NTRU LPRime, and BIKE), the plaintext is frequently fixed and the key is the payload. However, we set the plaintext as payload and set the key fixed in the experiment. This is because we are intended to conduct the experiment to validate the proposed attack in a manner more severe to the attacker, such that the evaluation becomes general for various modes of operation and masking implementation. For example, some masked AES implementation

Table 4: NN accuracy to estimate PRF input

	Non-protected AES/SHAKE s/w	Masked AES s/w	Masked AES h/w
Accuracy	1.000	0.996	0.703

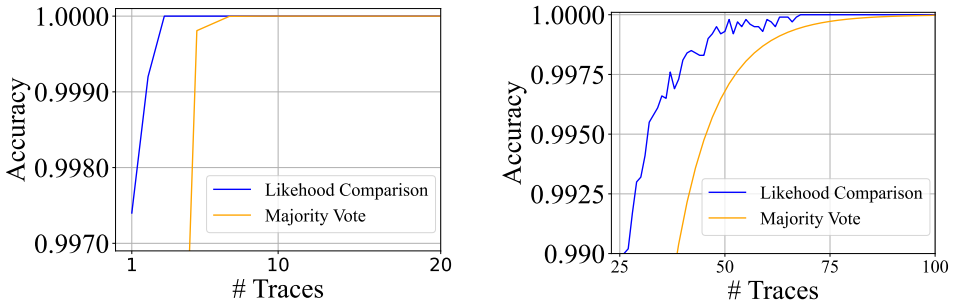
Masked bit-sliced AES software. We implement a first-order masked bit-sliced AES software provided in [git21], corresponding to Schwabe's and Stoffelen's paper [SS16]. We implement it on an STM32F407VGT6U equipped on an STM32F407G-DISC1 board as the source code was for the microcontroller, and we acquired the side-channel traces by measuring its EM emanation using an EM probe LANGER EMV-Technik RF-U 5-2. We used 40,000 traces for training and used 10,000 traces for validation among 40,000 traces. We also used another 10,000 traces for test. The plaintext and key are given in the manner same as the above.

Masked AES hardware based on TI. We implement a masked AES hardware based on first-order TI presented by Ueno *et al.* [UHA17] on a Xilinx Kintex-7 FPGA equipped with SAKURA-X board. First-order TI is one of the most promising countermeasures against first-order SCAs. The plaintext and key are given in the manner same as the above. We measure the supply voltage current as the side-channel traces through the coaxial connector on the SAKURA-X board. We used 1,010,000 traces for training and used 10,000 traces for validation among 1,010,000 traces. We also used another 10,000 traces for test. For the training and attack, we use logarithmic traces instead of raw traces in order to make it easier for the NN to be trained with regard to the higher-order moments as discussed in Appendix (In fact, we confirmed that the distinguish attack was not successful for the masked hardware when using raw traces). To derive the logarithmic traces, we firstly subtracted the smallest value of a trace from each sample point and then add 10^{-15} like a clipping value to each point such that all points become greater than zero, and finally we take the logarithm of each resulting point.

6.2 Accuracy evaluation

Table 4 reports the accuracy of trained NN for the test. Since the accuracy of "test" corresponds to that of the attack phase, the oracle accuracy can be evaluated according to the test accuracy. From Table 4, we can confirm that the trained NN achieve a sufficiently high accuracy to perform the proposed SCA. For the non-protected implementations, the NN model achieves a 100% accuracy for the test, which indicates that we can readily realize one oracle access from only one trace without any majority vote nor likelihood comparison. In contrast, the NN model achieves 99.6% and 70.3% accuracy for the masked bit-sliced software and TI-based masked hardware, respectively. It is more difficult to distinguish the input of the TI-based masked hardware than the mask software due to its lower SNR and/or advantage of TI. Still, the accuracy would be sufficiently high to perform the full-key recovery if implementing the plaintext-checking oracle with a majority vote or likelihood comparison.

(*e.g.*, masked hardware in [UHA17]) does not protect the key scheduling parts because it causes no DPA leakage, although it causes an exploitable leakage for the distinguish attack. Therefore, our experiment only aims at exploiting the leakage only from the round function part, which causes always an exploitable leakage independently of the mode of operation and masking implementation, for a general and severe evaluation. (More precisely, if the plaintext is a payload and key is fixed, only round function part is a leakage source but key scheduling is not, because the key scheduling part always processes an identical value for a fixed key. By contrast, both round datapath and key scheduling datapath are a leakage source for the distinguish attack, if the key is a seed and the plaintext is fixed. Thus, the experiment validates the proposed attack in a more general and severe manner, as the experiment is harder for the attacker.)



(a) Masked AES s/w (b) Masked AES h/w

Figure 1: Oracle accuracy with majority vote or likelihood comparison.

We then evaluate the oracle accuracy using the majority vote or likelihood comparison. Figure 1 illustrates the relation between oracle accuracy and number of traces for the protected implementations, where the horizontal axis denotes the number of traces and the vertical axis denotes the oracle accuracy for (a) masked AES software and (b) masked AES hardware. Here, the orange and blue curves denote the oracle realized with majority vote and likelihood comparison, respectively. Whereas the majority vote is evaluated in an analytical manner using Eq. (1) for odd numbers of traces, the likelihood comparison is evaluated experimentally using a shuffled test data repeatedly. More precisely, to evaluate the accuracy be the likelihood comparison, we repeat the following procedure 10,000 times: we randomly obtain 1,000 traces for reference plaintext or random plaintext from the test detaset, calculate the NLL in Eq. (2) for each hypothetical oracle output $b = 0$ or 1, and determine the oracle output as the smaller NLL. From Figure 1, we can confirm that the likelihood comparison would be more accurate and effective than majority vote according to the Neyman–Peason lemma. The distinguisher with a likelihood comparison can achieve a 100% test accuracy with at least 3 and 75 traces for masked software and hardware, respectively. In contrast, supposing that for example 10,000 oracle accesses are hypothetically required for key recovery, the majority vote requires at least 7 and 163 traces for a 99.999% success rate for masked software and hardware, respectively. In consequence, we can confirm that both methods can achieve a sufficient accuracy even when the implementation is masked.

6.3 Evaluation of number of traces for successful key recovery

Table 5 reports the number of side-channel traces required for a successful key recovery if using the side-channel distinguisher evaluated in the above, respectively. For masked implementations, we adopted the distinguisher based on the likelihood comparison with 3 and 75 traces according to the evaluation in Section 6.2. In Table 5, we suppose that the oracle realized by the side-channel distinguisher is completely accurate if it achieves a 100% accuracy at the test.

As a matter of course, we require more traces for key recovery if a larger number of traces is required to realize an accurate oracle. Therefore, the key recovery from masked implementation is more difficult than non-protected implementations. In particular, the attack on masked hardware based on TI requires 75 times more traces than non-protected implementations. Thus, we confirm a certain level of the effectiveness of masking countermeasures as they render the attack more difficult. However, our experimental result reveals that the attack is still feasible for the KEMs even if the PRF implementation is protected, regarding that the modern SCA evaluation is conducted with more-than 10M or 100M traces (*e.g.*, [SM15, SM19, SBM19]). In the sense, at least, the first-order masking countermeasures are not essential solution to counter the proposed SCA. The

Table 5: Number of side-channel traces required for successful proposed attack (partial-key recovery for BIKE and except for Classic McEliece)

KEM type	Scheme	Instance	# Side-channel traces for attack phase		
			Non-protected AES/SHAKE	Masked AES s/w	Masked AES h/w
Lattice	Kyber	Kyber-512	1,536	4,608	115,200
		Kyber-1024	3,072	9,216	230,400
	Saber	LightSaber-KEM	3,072	9,216	230,400
		FireSaber-KEM	3,072	9,216	230,400
	FrodoKEM	FrodoKEM-640	25,600	76,800	1,920,000
		FrodoKEM-1344	43,008	129,024	3,225,600
	NTRU Prime	ntrulpr653	1,306	3,918	97,950
		ntrulpr1277	2,554	7,662	191,550
		sntrup653	2,712	8,136	203,400
		sntrup1277	5,175	15,525	388,125
	NTRU	ntruhrss701	2,804	8,412	210,300
		ntruhps2048509	1,018	3,054	76,350
		ntruhps4096821	1,642	4,926	123,150
Code	HQC	hqc128	18,111	54,333	1,358,325
		hqc256	58,536	175,608	4,390,200
	BIKE	Level 1	3M	9M	225M
		Level 5	N/A	N/A	N/A
	Classic McEliece	Any	N/A	N/A	N/A
Isogeny	SIKE	SIKEp434	274	822	20,550
		SIKEp751	478	1,434	35,850

evaluation of the DL-based distinguish attack on higher-order masked implementation and the development of countermeasure are the important future work.

7 Conclusion

This paper presents a generic power/EM attack methodology using a plaintext-checking oracle on KEMs based on FO transformation and its variant. The proposed SCA exploits the side-channel leakage during PRF execution in re-encryption to realize a plaintext-checking oracle, namely, to distinguish whether the input to PRF is equalt the reference plaintext or not. We demonstrate that all KEMs in the NIST PQC third-round candidates except for Classic McEliece are vulnerable to the proposed attack. We also present a DL-based side-channel distinguisher design, and validate it through experimental attacks on various PRF implementations, including ones protected by a masking countermeasure. In consequence, we confirm that the proposed SCA can perform a key recovery of many KEM implementations, even if the PRF implementation is protected by a sophisticated countermeasure such as first-order TI.

The proposed attack has two significant advantage: the generality and applicability. First, the proposed attack realizes a plaintext-checking oracle through a side-channel leakage. Since many modern KEMs are known to be vulnerable to an adaptive attack using the plaintext-checking oracle, the proposed attack can be generally applied to these KEM schemes. Second, the proposed SCA does not require the detailed knowledge of target implementation and therefore can be applied to (relatively) black-box implementations. Thus, when implementing a KEM, we should be aware of the proposed attack if an adaptive attack on the underlying PKE is known and the application can be threatened by power/EM SCA.

In the scenario of SCA on KEM.Decaps, the attacker can perform a profiling using the target device itself without any secret key, which indicates that KEM implementations should be resistant to profiling attacks including DL-based ones, if SCA can be a threat for the application. Evaluation of higher-order masking against the proposed SCA with

DL-based distinguisher and developing an effective countermeasure are the important future work for realizing a secure KEM implementation. We are also planning to investigate the applicability of the proposed SCA to KEMs others than the NIST PQC third-round candidates.

Acknowledgment

This work has been supported by JSPS Kakanhi Grant No. 17H00729 and 19H21526, JST CREST No. JPMJCR19K5, and JST PRESTO No. JPMJPR18M3

Appendix: On the use of logarithmic trace in attacking masked hardware

The d-th order masked implementation is broken by an SCA using $(d+1)$ statistical moment according to the bounded momoent leakage model [BDF+17]. Here, we should exploit at least second-order statistical moment to break the first-order masked AES hardware. Here, the masked hardware only has a univariate higher-order leakage, because the hardware evaluates the shared function for an intermediate value simultaneously in one clock cycle, which is so-called share-parallel implementation. This implies that computing the $(d+1)$-th power (*i.e.*, squaring herein) of each point of centerized traces in advance would make it easier to break the masked AES hardware.

In this paper, for more efficient training and attack with less traces, we applied a logarithmic function to each point before the training and attack. The motivation behind the logarithmic function is that its combination with SELU, which is the activation function used in our NN, can efficiently compute the $(d+1)$-th power of each point of input traces. SELU outputs $\rho(\exp(x)-1)$ for a negative input x, where ρ is a positive coefficient. The NN used in the experiment employs a convolutional layer with a filter size of three as the first layer, and the SELU function is applied to the filter output of the layer. Since we apply a logirithmic function to the input trace, each point of traces is processed through a logarithmic function, weighted sum, and SELU function. Here, if the weight is given as a negative value $-w$ (where w is a positive real number), we equivalently compute the w-th negative power of input trace due to the logarithmic function and SELU as

$$\mathrm{SELU}(-w\log x) = \rho(\exp(-w\log x)-1) = \rho(x^{-w}-1).$$

Thus, applying a logarithmic function to input trace is equivalent to computing the w-th power for CNN with SELU. Note that the filter size of our CNN is given as three, and the SELU input should be given in a form of $w_0\log x_0 + w_1\log x_1 + w_2\log x_2$. Therefore, if one of the weights is trained as $w_j \geq d+1$ and other two weights are relatively negligible, the NN can approximately compute the $(d+1)$-th negative power and can extract the $(d+1)$-th univariate leakage. The proposed method based on logarithmic function requires a far less cost than an NN is trained to mimic a nonlinear function like the $(d+1)$-th power. Thus, the proposed method would be effective in attacking a masked hardware with univariate higher-order leakage.

References

[A+20] Erdem Alkim et al. FrodoKEM—practical quantum-secure key encapsulation from generic lattices. https://frodokem.org, 2020.

[BDF+17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations

of masking schemes and the bounded moment leakage model. In *Advances in Cryptology—Eurocrypt 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–556, 2017.

[BDH+21] Shivam Bhasin, Jan-Pieter D'Anvers, Daniel Heinz, Thomas Pöppelmann, and Michiel Van Beirendonck. Attacking and defending masked polynomial comparison for lattice-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021:XXX–XXX, 2021.

[BDL+19] Ciprian Băetu, F. Betül Durak, Huguenin-Dumittan Loïs, Abdullah Talayhan, and Serge Vaudenay. Misuse attacks on post-quantum cryptosystems. In *Advances in Cryptology—Eurocrypt 2019*, volume 11477 of *Lecture Notes in Computer Science*, pages 747–776, 2019.

[BHH+19] Nina Bindel, Mike Hamburg, Kathrin Hövelmanns, Andreas Hülsing, and Edoardo Persichetti. Tighter proofs of CCA security in the quantum random oracle model. In *Theory of Cryptography*, volume 11892 of *Lecture Notes in Computer Science*, pages 61–90, 2019.

[BPO+20] Florian Bache, Clara Paglialong, Tobias Oder, Tobias Schneider, and Tim Güneysu. High-speed masking for polynomial comparison in lattice-based KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020:483–507, 2020.

[BPS+18] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. IACR ePrint archive: Report 2018/053, 2018. https://eprint.iacr.org/2018/053.

[DDS+19] Jintai Ding, Joshua Deaton, Kurt Schmidt, Vishakha, and Zheng Zhang. A simple and efficient key reuse attack on NTRU cryptosystem. IACR ePrint archive: Report 2019/1022, 2019. https://eprint.iacr.org/2019/1022.

[DV21] Orsini Emmanuela D'Anvers, Jan-Pieter and Frederik Vercauteren. Error term checking: Towards chosen ciphertext security without re-encryption. IACR ePrint archive: Report 2021/080, 2021. https://eprint.iacr.org/2021/080.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology—CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Seience*, pages 537–554, 1999.

[git21] Fast, constant-time and masked AES assembly implementations for ARM Cortex-M3 and M4. https://github.com/Ko-/aes-armcortexm, May 2021.

[GJS16] Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 789–815, 2016.

[GPST16] Steven D. Galbraith, Christophe Petit, Barak Shani, and Bo Yan Ti. On the security of supersingular isogeny cryptosystems. In *Advances in Cryptology—ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Seience*, pages 63–91, 2016.

[GTN20] Qian Guo, Johansson Thomas, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the Fujisaki–Okamoto transformation and its application on FrodoKEM. In *Advances in Cryptology—CRYPTO '20*, volume 12171 of *Lecture Notes in Computer Seience*, pages 359–386, 2020.

[HHK17] Denis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki–Okamoto transformation. In *Theory of Cryptography*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, 2017.

[HS99] Jeffrey Hoffstein and Joseph H. Silverman. Reaction attacks against the NTRU public key cryptosystem. NTRU Technical Report, 1999. Available at https://ntru.org/resources.shtml.

[HV20] Loïs Huguenin-Dumittan and Serge Vaudenay. Classical misuse attacks on NIST round 2 PQC - the power of rank-based schemes. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *Applied Cryptography and Network Security - 18th International Conference, ACNS 2020, Rome, Italy, October 19-22, 2020, Proceedings, Part I*, volume 12146 of *Lecture Notes in Computer Science*, pages 208–227. Springer, 2020.

[J+20] David Jao et al. SIKE—Supersingular Isogeny Key Encapsulation. https://sike.org, 2020.

[JDF11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Proceedings in PQCrypto 2011*, volume 7071 of *Lecture Notes in Computer Science*, pages 19–34, 2011.

[JJ00] Éliane Jaulmes and Antoine Joux. A chosen-ciphertext attack against NTRU. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2000.

[KAJ17] Brian Koziel, Reza Azarderakhsh, and David Jao. Side-channel attacks on quantum-resistant supersingular isogeny Diffie–Hellman. In *Selected Areas in Cryptography—SAC 2017*, volume 10719 of *Lecture Notes in Computer Science*, pages 64–81, 2017.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology—CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[KPH+19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019:148–179, 2019.

[KPP20] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attack on Keccak. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020:243–268, 2020.

[KRSS19] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. IACR ePrint archive: Report 2019/844, 2019. https://eprint.iacr.org/2019/844.

[LNPS20] Norman Lahr, Ruben Niedarhgen, Richard Petri, and Simona Samardjiska. Side channel information set decoding using iterative chunking: Plaintext recovery from the "Classic McEliece" hardware reference implementation. In *Advances in Cryptology—ASIACRYPT 2020*, volume 12491 of *Lecture Notes in Computer Seience*, pages 881–910, 2020.

[MTSB13] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*, pages 2069–2073. IEEE, 2013.

[MWM19] Thorben Moos, Ferix Wegener, and Amir Moradi. DL-LA: Deep learning leakage assessment—a modern roadmap for SCA evaluations. IACR ePrint archive: Report 2019/505, 2019. https://eprint.iacr.org/2019/505.

[NDGJ21] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johanson. A side-channel attack on a masked IND-CCA secure Saber KEM. IACR ePrint archive: Report 2021/079, 2021. https://eprint.iacr.org/2021/079.

[NIS20] NIST. Post-quantum cryptogprahy. https://csrc.nist.gov/projects/post-quantum-cryptography, 2020.

[NP33] Jerzy Neyman and Egon Sharpe Peason. IX. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society A*, 231:694–706, 1933.

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2–secure and masked ring-LWE implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018:142–174, 2018.

[OUKT21] Yuki Osumi, Shusaku Uemura, Momonari Kudo, and Tsuyoshi Takagi. Key mismatch attack on SABER. In *SCIS 2021*, January 2021. In Japanese.

[PCP20] Guilherme Perin, Łukasz Chmielewski, and Stjepan Picek. Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 337–364, 2020.

[PP21] Peter Pessl and Lukas Prokop. Fault attacks on CCA-secure lattice KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021:37–60, 2021.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In *International Conference on Cryptographic Hardware and Embedded Systems*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–553. Springer, 2017.

[pqm21] Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4, April 2021.

[RBRC20] Prasanna Ravi, Shivam Bhasin, Sujoy Sinha Roy, and Anupam Chattopadyay. On exploiting message leakage in (few) NIST PQC candidates for practical message recovery and key recovery attacks. IACR ePrint archive: Report 2020/1559, 2020. https://eprint.iacr.org/2020/1559.

[REB+21] Prasanna Ravi, Martianus Frederic Ezerman, Shivam Bhasin, Anupam Chattopadhyay, and Sujoy Sinha Roy. Generic side-channel assisted chosen-ciphertext attacks on Streamlined NTRU Prime. IACR ePrint archive: Report 2021/718, 2021. https://eprint.iacr.org/2021/718.

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020:307–335, 2020.

[SBM19] Aein Rezaei Shahmirzadi, Dušan Božilov, and Amir Moradi. New first-order secure AES performance records. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, pages 304–327, 2019.

[SKC+19] Bo-Yeon Sim, Jihoon Kwon, Kyu Young Choi, Jihoon Cho, Aeson Park, and Dong-Guk Han. Novel side-channel attacks on quasi-cyclic code-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019:180–212, 2019.

[SKL+20] Bo-Yeon Sim, Jihoon Kwon, Joohoo Lee, Il-Ju Kim, Tae-Ho Lee, Hyojin Yoon, Jihoon Cho, and Dong-Gak Han. Single-trace attacks on message encoding in lattice-based KEMs. *IEEE Access*, 8:183175–183191, 2020.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 30–46. Springer, 2005.

[SM15] Tobias Schneider and Amir Moradi. Leakage assesment methodology—A clear roadmap for side-channel evaluations. In *Workshop on Cryptographic Hardware and Embedded Systems*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.

[SM19] Aein Rezaei Shahmirzadi and Amir Moradi. Re-consolidating first-order masking schemes—nullifying fresh randomness. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019:123–145, 2019.

[SS16] Peter Schwave and Ko Stoffelen. All the AES you need on Cortex-M3 and M4. In *Selected Areas in Cryptography—SAC 2016*, volume 10532 of *Lecture Notes in Computer Science*, pages 180–194, 2016.

[SXY18] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In *Advances in Cryptology—EUROCRYPT 2018*, volume 10822 of *Lecture Notes in Computer Science*, pages 520–551. Springer, 2018.

[UHA17] Rei Ueno, Naofumi Homma, and Takafumi Aoki. Toward more efficient DPA-resistant AES hardware architecture based on threshold implementation. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, volume 10348 of *Lecture Notes in Computer Science*, pages 50–64, 2017.

[vBDK+21] Michiel van Beirendonck, Jan-Peter D'anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel-resistant implementation of SABER. *ACM Journal on Emerging Technologies on Computing Systems*, 17(2), 2021.

[WAGP20] Lennert Wouters, Victors Arribas, Benedikt Gierlichs, and Bart Praneel. Revisiting a methodology for efficient CNN architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020:147–168, 2020.

[XIU+21] Keita Xagawa, Akira Ito, Rei Ueno, Junko Takahashi, and Naofumi Homma. Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates. IACR ePrint archive: Report 2021/840, 2021. https://eprint.iacr.org/2021/840.

[XPRO20] Zhuang Xu, Owen Pemberton, Sujoy Sinha Roy, and David Oswald. Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of Kyber. IACR ePrint archive: Report 2020/912, 2020. https://eprint.iacr.org/2020/912.

[YZ17] Yu Yu and Jiang Zhang. Lepton. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

[ZBHV20] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020:1–36, 2020.

[ZCQD21] Xiaohan Zhang, Chi Cheng, Yue Qin, and Ruoyu Ding. Small leaks sink a great ship: An evaluation of key reuse resilience of PQC third round finalist NTRU-HRSS. IACR ePrint archive: Report 2021/168, 2021. https://eprint.iacr.org/2021/168.

[ZYD+20] Fan Zhang, Bolin Yang, Xiaofei Dong, Sylvain Guilley, Zhe Liu, Wei He, Fangguo Zhang, and Kui Ren. Side-channel analysis and countermeasure design on ARM-based quantum-resistant SIKE. *IEEE Transactions on Computers*, 69:1681–1693, 2020.