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 Social Informatics 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 the Fujisaki–Okamoto (FO) transformation and its variants. The FO transformation has been widely used in actively securing 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 during execution of a pseudorandom function (PRF) or pseudo-random number generator (PRG) in the re-encryption of KEM decapsulation as a *plaintext-checking oracle* that tells whether the PKE decryption result is equivalent to the *reference* plaintext. The generality and practicality of the plaintext-checking oracle allow the proposed attack to attain a full-key recovery of various KEMs when an active attack on the underlying PKE is known. This paper demonstrates that the proposed attack can be applied to most NIST PQC third-round KEM candidates, namely, Kyber, Saber, FrodoKEM, NTRU, NTRU Prime, HQC, BIKE, and SIKE (for BIKE, the proposed attack achieves a partial key recovery). The applicability to Classic McEliece is unclear because there is no known active attack on this cryptosystem. This paper also presents a side-channel distinguisher design based on deep learning (DL) for mounting the proposed attack on practical implementation without the use of a profiling device. The feasibility of the proposed attack is evaluated through experimental attacks on various PRF implementations (a SHAKE software, an AES software, an AES hardware, a bit-sliced masked AES software, and a masked AES hardware based on threshold implementation). Although it is difficult to implement the oracle using the leakage from the TI-based masked hardware, the success of the proposed attack against these implementations (even except for the masked hardware), which include masked software, confirms its practicality.

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. As it is usually difficult to construct a chosen ciphertext attack (CCA)-secure PKE, the Fujisaki–Okamoto (FO) transformation [FO99] and its variants (*e.g.*, [HHK17,

SXY18, BHH+19]) have been commonly used to produce CCA-secure key encapsulation mechanisms (KEMs) from a chosen plaintext attack (CPA)-secure PKE *via re-encryption* and *equality checking*; most KEM schemes in NIST post-quantum cryptography (PQC) competition [NIS20] follow this CPA-to-CCA transformation approach. Although the theoretical security of such KEM schemes has been extensively analyzed, side-channel analysis (SCA), a type of attack on cryptographic implementation using side-channel leakage (*e.g.*, execution time, power consumption, and electromagnetic (EM) emanation), can potentially 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 KEMs have mainly focused on the decryption of the underlying PKE with the goal of recovering the secret key (*e.g.*, [PPM17, ZYD+20]). In contrast, some recent studies have shown that the FO transformation can leak the secret key, even if the underlying PKE is securely implemented [GTN20, RRCB20, PP21]. These attacks exploit side-channel leakage or fault injection to obtain information about the PKE decryption result, and then mount a chosen-ciphertext attack on the underlying PKE. In fact, such side-channel-assisted chosen-ciphertext attacks have been studied on public key primitives after the disclosure of Bleichenbacher's padding oracle attack on RSA PKCS [Ble98]. Focusing on post-quantum KEMs based on the FO transformation, Guo *et al.* [GTN20] present 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 the 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 the FO transformation in code- nor isogeny-based KEM(s) (*e.g.*, HQC, BIKE, and SIKE). Thus, a detailed evaluation of the applicability/limitations of SCAs on the 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 CCA security for most KEM 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 as follows:

- We present a generic power/EM SCA methodology for KEMs based on the FO transformation and its variants. The key idea underlying the proposed attack is the creation of a *plaintext-checking oracle* through a side-channel trace to mount a chosen-ciphertext attack on the underlying CPA-secure PKE. This oracle tells whether or not the PKE decryption result in decapsulation is equivalent to the *reference* plaintext. The reference plaintext means the PKE decryption result corresponding to a valid *reference* ciphertext. To realize the oracle, the proposed attack exploits side-channel leakage during execution of pseudorandom function (PRF) or pseudorandom number generator (PRG) in re-encryption of the KEM decapsulation. This allows it to distinguish whether or not the PKE decryption result is the fixed reference plaintext. The proposed SCA focusing on the PRF leakage can be also used to create a *decryption failure oracle*, which is another major oracle in attacking lattice-based KEMs, as the attack on Streamlined NTRU Prime shown in this paper (following the attack in [REB+21]). As such, the proposed attack can be performed even if the underlying PKE implementation has no secrecy leakage. As many PKEs are vulnerable to adaptive chosen-ciphertext attacks, the proposed attack can be widely applied to many KEMs including lattice-, code-, and isogeny-based KEMs.

Table 1: Applicability of implementation attacks focusing on FO transformation to NIST PQC third-round KEM candidates and their potential countermeasure/mitigation

		[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	No	No	No	**Yes**
	NTRU Prime	Partially 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/mitigation		Constant-time	Redundancy	Masking	

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

- We investigate the applicability of the proposed attack to NIST PQC third-round KEM candidates (four finalists and five alternatives), and demonstrate that Kyber, Saber, FrodoKEM, NTRU, NTRU Prime, HQC, BIKE, and SIKE are vulnerable to the proposed SCA. The proposed attack achieves a partial key recovery of BIKE. Its applicability to Classic McEliece remains unclear as no adaptive chosen-ciphertext attack on Classic McEliece is known. The applicability of the proposed and conventional attacks are summarized in Table 1, in which the generality of the proposed attack—one of major advantages over conventional attacks—is confirmed. We stress here that this paper is the first report on power/EM analysis on the FO transformation of code- and isogeny-based KEMs, although some SCAs on the FO transformation of lattice-based KEMs have been already discussed in previous works, which are not fully generalized (*e.g.*, [RRCB20] on Kyber, Saber, and FrodoKEM).

- We present a deep-learning (DL)-based distinguisher for implementing the plaintext-checking oracle, which is designed as a two-classification neural network (NN), and allows for attacks without specific assumption nor knowledge about the target implementation. In addition, we also describe how to realize the distinguisher with a convincing accuracy using an NN model whose accuracy is insufficient (in many cases, NN model accuracy for SCA can be low owing to the presence of noise and/or SCA countermeasures [ISUH21]). Thus, as demonstrated in this paper, the proposed NN-based distinguisher can be used to attack practical implementations in a black-box manner even when an SCA countermeasure such as masking are implemented. Note that the proposed attack requires no profiling device for acquiring a training dataset, since it is acquired from the target implementation under our scenario, 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 experiments, we target five implementations: a non-protected SHAKE and AES software obtained from an open-source cryptographic softwares library `pqm4` [KRSS19, pqm21], an open-source AES hardware developed for side-channel standard attack evaluation board (SASEBO) [Toh], an open-source masked bit-sliced 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. Our results confirm that the NN model can achieve a sufficiently high test accuracy to perform the key recovery attack even for masked software

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

KeyGen

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** $(\mathsf{sk}, \mathsf{pk}, s)$;

Encaps

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

Decaps

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: $r' \leftarrow \mathsf{G}(m'[, \mathsf{pk}])$;
4: $c' \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m'; r')$;
5: **if** $c = c'$ **then**
6: **return** $\mathsf{H}(m', c)$;
7: **else**
8: **return** $\mathsf{H}_{\mathsf{prf}}(s, c)$;

implementations, whereas it is difficult to break the masked TI-based hardware in our environment. Finally, we rigorously and comprehensively evaluate the number of side-channel traces required for a successful key recovery to demonstrate the practicality of the proposed SCA on the post-quantum KEMs.

- The source codes used in our experiment are publicly available at https://github.com/ECSIS-lab/curse_of_re-encryption.

1.3 Paper organization

The remainder of this paper is organized as follows. Section 2 reviews KEMs based on the FO transformation and the previous timing and power/EM SCAs on KEMs focusing on the FO transformation. Section 3 describes the proposed SCA methodology on the basis of a plaintext-checking oracle realized *via* side-channel leakage. Section 4 demonstrates the application of the proposed attack to NIST PQC third-round KEM candidates. Section 5 presents the side-channel distinguisher design for mounting the proposed attack on practical implementations 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 the 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 obtained using a CPA-secure PKE with the FO transformation or its variant (*e.g.*, [HHK17, SXY18, BHH+19]); most NIST PQC KEM candidates follow this CPA-to-CCA transformation approach.

Algorithm 1 illustrates $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ based on a (standard) FO transformation, where PKE is a CPA-secure probabilistic PKE comprising a key generation algorithm Gen, an encryption algorithm Enc, and a decryption algorithm Dec. Here, as a major example, we consider a KEM that returns a pseudorandom number instead of a rejection symbol $\perp$ in the case of an invalid ciphertext, which implies *implicit rejection*. Given a security parameter 1^λ, KEM.KeyGen first generates a key pair $(\mathsf{sk}, \mathsf{pk})$ using the PKE key generation PKE.Gen. Then, s is generated as a random plaintext of the PKE from the message space $\mathcal{M}$ at Line 3. Finally, the algorithm returns the triplet $(\mathsf{sk}, \mathsf{pk}, s)$.

KEM.Encaps first randomly generates a message m from a message space $\mathcal{M}$ and then evaluates a random oracle G on m or on a pair of m and pk (*e.g.*, in the cases of BIKE and SIKE, respectively), which is usually realized using a PRF/PRG. In Line 4, the PKE encryption PKE.Enc is performed using a public key pk, message m, and randomness r. Then, a random oracle H is evaluated on m and c to derive the shared secret k. The

input format and output length of those random oracles are determined in accordance with each KEM specification, as summarized in Table 2. Finally, the algorithm returns the ciphertext c corresponding to k. Note that the ciphertext c may be a tuple.

$\mathsf{KEM.Decaps}$ first performs the PKE decryption for c using the secret key sk to obtain the plaintext m'. Then, analogously to $\mathsf{KEM.Encaps}$, $\mathsf{KEM.Decaps}$ generates r' as $\mathsf{G}(m')$ or $\mathsf{G}(m', \mathsf{pk})$, and evaluates $\mathsf{PKE.Enc}(\mathsf{pk}, m'; r')$. This procedure is called *re-encryption*. At Line 5, the $\mathsf{KEM.Decaps}$ algorithm performs *equality checking*, namely, examines whether the re-encryption result c' is equal to the ciphertext c. If $c = c'$, the algorithm returns the shared secret $k = \mathsf{H}(m', c)$ as the ciphertext is valid; otherwise, the algorithm returns a pseudorandom number of $\mathsf{H}_{\mathsf{prf}}(s, c)$ (instead of $\perp$) as the ciphertext is invalid, where $\mathsf{H}_{\mathsf{prf}}$ is another random oracle or equivalent to H. Thus, the KEM scheme gives any active attacker no information about the PKE decryption result for invalid ciphertext.

In many modern KEM schemes, G, H, and $\mathsf{H}_{\mathsf{prf}}$ are instantiated using SHAKE or $\mathsf{SHA3}$. There are some variants of the 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/PRG, re-encryption, or equality/validity check). Although some variants avoid the complete re-encryption for computational efficiency (*e.g.*, [DOV21] and NTRU submitted to NIST PQC), the proposed SCA would be applicable to these CPA-to-CCA-secure transformations as long as they involve PRF/PRG and/or procedure corresponding to equality/validity check. Finally, we summarize how KEMs implement G, H, $\mathsf{H}_{\mathsf{prf}}$, and F, which is used to compute an additional hash added into a ciphertext, in Table 2.

2.2 Previous SCAs on FO transformation

2.2.1 Timing analysis

Guo *et al.* [GTN20] present an SCA focusing on the FO transformation. The attack utilizes a timing side-channel to realize a plaintext-checking oracle for lattice- and code-based KEMs. As the timing attack exploits the equality check between the ciphertext and re-encryption result (*i.e.*, Line 5 in $\mathsf{KEM.Decaps}$ of Algorithm 1) rather than $\mathsf{PKE.Dec}$, the attack can be applied to constant-time PKE implementation, unless overall 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 adaptive 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. The timing attack implements the plaintext-checking oracle *via* a timing side-channel. The attacker generates an invalid ciphertext $c' = c + \delta$ where δ is determined according to the adaptive attack. For lattice- and code-based PKEs, if δ is sufficiently small for the underlying scheme, the PKE decryption result for c' will be equivalent to m, which indicates that the re-encryption result should be c in this case. Otherwise, the PKE decryption result will be a random plaintext $\hat{m}$, which will be re-encrypted to a random ciphertext $\hat{c}$ that differs significantly from c. Then, $\mathsf{PKE.Decaps}$ performs the equality check, namely, compares the ciphertext $c + \delta$ with the re-encryption result. Here, ciphertext of lattice- and code-based PKEs is treated as long vectors 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

Table 2: Summary of variants of FOs in NIST PQC Round 3 KEM Candidates (finalists and alternates): Before version 4.2, BIKE's G uses SHA384 and AES256-CTR. For a function Hash, $\mathsf{Hash}_\ell(x)$ will output the first ℓ bits of $\mathsf{Hash}(x)$. $\mathsf{SHA3\text{-}512}_r$ and $\mathsf{SHA3\text{-}512}_l$ output the first and second 256 bits of SHA3-512. FrodoKEM-640 uses $(\mathsf{SHAKE}, k) = (\mathsf{SHAKE128}, 128)$, FrodoKEM-976 uses $(\mathsf{SHAKE}, k) = (\mathsf{SHAKE256}, 192)$, and FrodoKEM-1344 use $(\mathsf{SHAKE}, k) = (\mathsf{SHAKE256}, 256)$. In SIKE, e_2 and k are parameters. BIKE and SIKE additionally use L in the underlying PKE to mask a message with masking value computed from the shared random value L(shared). BIKE uses $\mathsf{SHA3\text{-}384}_{256}(r)$ and SIKE uses $\mathsf{SHAKE256}_n(j)$ as L.

Name	G	F
Classic McEliece	–	$\mathsf{SHAKE256}_{256}(\mathtt{0x02}, m)$
Kyber	$\mathsf{SHA3\text{-}512}_r(m, \mathsf{SHA3\text{-}256}(\mathsf{pk}))$ [a]	–
NTRU	–	–
Saber	$\mathsf{SHA3\text{-}512}_r(m, \mathsf{SHA3\text{-}256}(\mathsf{pk}))$ [b]	–
BIKE	$\mathsf{SHAKE256}(m)$	–
FrodoKEM	$\mathsf{SHAKE}_{2k,l}(\mathsf{SHAKE}_k(\mathsf{pk}), m)$ [c]	–
HQC	$\mathsf{SHAKE256}_{512}(m, \mathtt{0x03})$ [d]	$\mathsf{SHAKE256}_{512}(m, \mathtt{0x04})$
Streamlined NTRU Prime	–	$\mathsf{SHA512}_{256}(\mathtt{0x02}, \mathsf{SHA512}_{256}(\mathtt{0x03}, m), \mathsf{SHA512}_{256}(\mathtt{0x04}, \mathsf{pk}))$
NTRU LPRime	$\mathsf{SHA512}_{256}(\mathtt{0x05}, m)$ [e]	$\mathsf{SHA512}_{256}(\mathtt{0x02}, m, \mathsf{SHA512}_{256}(\mathtt{0x04}, \mathsf{pk}))$
SIKE	$\mathsf{SHAKE256}_{e_2}(m, \mathsf{pk})$	–
Name	H	$\mathsf{H}_{\mathsf{prf}}$
Classic McEliece	$\mathsf{SHAKE256}_{256}(\mathtt{0x01}, m, (c_0, c_1))$	$\mathsf{SHAKE256}_{256}(\mathtt{0x00}, s, (c_0, c_1))$
Kyber	$\mathsf{SHAKE256}_X(\mathsf{SHA3\text{-}512}_l(m, \mathsf{SHA3\text{-}256}(\mathsf{pk})), \mathsf{SHA3\text{-}256}(c))$	$\mathsf{SHAKE256}_X(s, \mathsf{SHA3\text{-}256}(c))$
NTRU	$\mathsf{SHA3\text{-}256}(m)$	$\mathsf{SHA3\text{-}256}(s, c)$
Saber	$\mathsf{SHA3\text{-}256}(\mathsf{SHA3\text{-}512}_l(m, \mathsf{SHA3\text{-}256}(\mathsf{pk})), \mathsf{SHA3\text{-}256}(c))$	$\mathsf{SHA3\text{-}256}(s, \mathsf{SHA3\text{-}256}(c))$
BIKE	$\mathsf{SHA3\text{-}384}_{256}(m, c)$	$\mathsf{SHA3\text{-}384}_{256}(s, c)$
FrodoKEM	$\mathsf{SHAKE}_k(c, \mathsf{SHAKE}_{2k,r}(\mathsf{SHAKE}_k(\mathsf{pk}), m))$	$\mathsf{SHAKE}_k(c, s)$
HQC	$\mathsf{SHAKE256}_{512}(m, c, \mathtt{0x05})$	–
Streamlined NTRU Prime	$\mathsf{SHA512}_{256}(\mathtt{0x01}, \mathsf{SHA512}_{256}(\mathtt{0x03}, m), c)$	$\mathsf{SHA512}_{256}(\mathtt{0x00}, \mathsf{SHA512}_{256}(\mathtt{0x03}, s), c)$
NTRU LPRime	$\mathsf{SHA512}_{256}(\mathtt{0x01}, m, c)$	$\mathsf{SHA512}_{256}(\mathtt{0x00}, s, c)$
SIKE	$\mathsf{SHAKE256}_k(m, c)$	$\mathsf{SHAKE256}_k(s, c)$

[a] Kyber uses an intermediate PKE scheme with short randomness which internally uses PRF $\mathsf{SHAKE256}_X(r, i)$ for $i = 1, 2, \ldots$ with appropriate length parameter X.
[b] Saber uses an intermediate PKE scheme with short randomness which internally uses XOF $\mathsf{SHAKE128}(r)$.
[c] FrodoKEM uses an intermediate PKE scheme with short randomness which internally uses XOF $\mathsf{SHAKE}(\mathtt{0x96}, r)$.
[d] HQC uses an intermediate PKE scheme with short randomness which internally uses XOF $\mathsf{SHAKE256}(r, \mathtt{0x02})$.
[e] NTRU LPRime uses an intermediate PKE scheme with short randomness which internally uses XOF $\mathsf{AES256\text{-}CTR}(r)$.

of KEM is achieved by repeated accesses to the plaintext-checking oracle for different δ's.

In [GTN20], Guo *et al.* demonstrate the application of this attack to FrodoKEM. Although the signal-to-noise ratio (SNR) of side-channel measurement (*i.e.*, accuracy of the oracle) would be problematic, the result indicates that the full-key recovery is sufficiently feasible. Since the disclosure of this attack, many PQC implementations have employed a fully constant-time conditional move (*e.g.*, `cmov`) for the secure 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 adaptive attack on SIKE.PKE uses invalid ciphertext(s) that differs significantly from reference ciphertext, indicating 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.* further note that their timing analysis may be carried out using a power/EM side-channel, because each pair of similar ciphertexts will have 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; the feasibility of such an attack is not validated.

2.2.2 Power/EM analysis

Ravi *et al.* [RRCB20] show an SCA on lattice-based KEMs. The attack is a side-channel-assisted CCA, ciphertexts of which are generated such that the decrypted (or decoded) plaintext is either 0 or 1 depending on a partial key. Here, the attacker cannot directly observe the plaintext due to the FO transformation; however, the side-channel leakage during re-encryption can be exploited to distinguish whether the plaintext is 0 or 1, which allows the attack to estimate the partial key. The attacker can recover the full key of some lattice-based KEMs by repeatedly querying the invalid ciphertexts to obtain different partial keys. In [RRCB20], Ravi *et al.* show that this methodology is applicable to six lattice-based KEMs, namely, Kyber, Saber, FrodoKEM, Round5, NewHope, and LAC. Ravi *et al.* also present a side-channel distinguisher based on a combined application of Welch's t-test and reduced template that yields a sufficiently feasible full-key recovery. Their distinguisher does not require the detailed knowledge of the target implementation.

Recently, Bhasin *et al.* [BDH+21] report SCA vulnerabilities of masked polynomial comparison schemes [OSPG18, BPO+20] for ciphertext equality check in lattice-based KEMs, and demonstrate 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 comprised of t-test like the test vector leakage assessment (TVLA) [SM15]. Note that, although the attack utilizes a plaintext-checking oracle as well as our SCA, the literature primarily studies the (in)security of masked polynomial comparison for lattice-based KEMs, and discusses only some lattice-based KEMs (*i.e.*, Kyber, Saber, and FrodoKEM). In this sense, the contributions and goal of this paper are different from those of [BDH+21], as this paper primarily 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 that is generally applicable to various PRF implementations.

In addition, extended CCA SCA approaches to lattice-based KEMs have been presented in [XPRO20, RBRC20, SKL+20, REB+21], and, Ngo *et al.* [NDGJ21] present 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 traces) because they employ chosen ciphertexts which result in more side-channel-leaky plaintext regarding features and implementation of the underlying PKE. 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 the FO transformation. In other words, these attacks achieve a higher efficiency by focusing on a scheme/implementation-specific aspect; therefore, they are less general in terms of KEM based on the FO transformation.

In addition to the approaches outlined above, there are some SCAs for code- and isogeny-based KEMs (*e.g.*, [SKC+19, LNPS20] and [KAJ17, ZYD+20], respectively). However, these attacks focus on the underlying PKE rather than the FO transformation. For code- and isogeny-based KEMs, no SCA focusing on the FO transformation is known.

As another attack direction, Kannwischer *et al.* [KPP20] present a single-trace SCA on SHA3 that recovers the secret input to SHA3 *via* a belief propagation based method, which is called soft-analytical SCA (SASCA). Although their attack is powerful, its feasibility heavily depends on the word length of the processor, the key length (*i.e.*, the input bits to be recovered), and the 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, SASCA requires the detail of implementation and can be prevented/mitigated using a common SCA countermeasure (*e.g.*, masking). Note that Kannwischer *et al.* show that their attack can be used for recovering the shared secret of KEMs, but do not show the secret key recovery.

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 engaged in a plaintext-checking oracle is called key-recovery plaintext-checking attack (KR-PCA). In this paper, we refer to "adaptive attack" as adaptive chosen-ciphertext attack.

For a given KEM, let c be a valid ciphertext named the *reference* ciphertext, and let m be the corresponding plaintext named the *reference* plaintext. Note here that m denotes the PKE decryption result, rather than the output of KEM.Decaps. The attacker can obtain a reference ciphertext corresponding to any reference plaintext by performing an encapsulation without secret key. 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' is equal to either the reference plaintext m or other plaintext $\hat{m}$. Formally, the 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 the FO transformation.

3.2 Proposed SCA

The proposed attack enacts a plaintext-checking oracle (or other abstracted decryption oracle such as decryption failure 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 obtains the side-channel leakage during PRF execution of PKE.Decaps for the reference ciphertext c. The attacker then 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. By contrast, if c' is decrypted to other plaintext $\hat{m}$, the two side-channel leakages should be meaningfully different. Thus, the attacker can distinguish whether or not the PKE decryption result is a reference plaintext from the side-channel leakage of PRF. As the proposed attack focuses on PRF leakage, it can perform a key recovery independently of the PKE.Dec implementation, even if it has no secrecy leakage.

The proposed SCA comprises a profiling phase and an attack phase. In the profiling phase, the attacker trains a classification model that uses side-channel trace(s) to distinguish which the PRF/PRG input is the reference plaintext or other random plaintext to enact the plaintext-checking oracle as mentioned above. In this paper, this trained model is called a side-channel distinguisher. In the attack phase, the attacker performs an adaptive attack on the underlying PKE using the trained distinguisher as the plaintext-checking oracle. Note that, although the attack employs a profiling phase, it does not require any profiling device because the profiling is performed using the target device without knowing the secret key, as in previous SCAs on lattice-based KEMs [RRCB20,XPRO20,RBRC20,SKL+20,NDGJ21]. The proposed SCA also does not require details on the target implementation, as DL enables us to train a model without leakage assumption nor specific knowledge about the target implementation. Such a DL-based side-channel distinguisher would be suitable to two classification of traces for fixed vs. random input, as Moos *et al.* show an efficient DL-based leakage assessment [MWM21].

4 Application to Post-Quantum KEMs

4.1 Lattice-based KEMs

4.1.1 Attack concept

To describe the underlying idea on KR-PCA on several prominent lattice-based PKEs, we consider a lattice-based PKE with a simplified notation. Suppose that, in the PKE decryption, the plaintext before decoding is given in the form of $\mathsf{Encode}(m) + ke + e'$, where k is the secret key, e and e' are errors, and 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 $\hat{m}$. Lattice-based PKEs are usually designed so that the decryption failure probability is negligibly small for valid ciphertext.

In a KR-PCA, the attacker queries a modified ciphertext $c' = c + \delta$ to the decryption oracle, where δ is an error added to the ciphertext. The modified ciphertext is decrypted to $\mathsf{Encode}(m) + ke + e' + \delta$ before decoding, where $ke + e' + \delta$ is the noise to be removed. Let m' be the decoded plaintext. 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 can determine $ke+e'+\delta$ through adaptive queries to the plaintext-checking oracle to find a value of δ such that $ke + e' + \delta = \gamma$. Thus, the attacker solves the linear equation to recover the secret key k because e, e', δ, and γ are available to the attacker. Furthermore, the number of oracle accesses needed for a full-key recovery can be reduced by querying a dedicated ciphertext, as mentioned in [BDL+19] and described in the following sections.

4.1.2 FrodoKEM

We herein describe the KR-PCA on FrodoKEM in [GTN20] as a representative and simple case. For the simplicity, we omit the detailed descriptions of attacks on Kyber and Saber because they are broken in a manner similar to FrodoKEM, as described in Section 4.1.3. Some instances of NTRU and NTRU Prime are also broken in a similar manner (we omit the detailed description for them as well), and 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 the secret key. 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

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

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 a KR-PCA, 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 component $\mathbf{E}\mathbf{S}' - \mathbf{E}'\mathbf{S} + \mathbf{E}'' + \Delta$ 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 a Δ such that $\Gamma = Q$ by adaptively

Algorithm 2 Key-recovery plaintext-checking attack on FrodoKEM

Input: Reference ciphertext (c_0, c_1), reference plaintext 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: $\quad\Delta \leftarrow \mathrm{ZeroMatrix}(n, \bar{n})$;
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: $\quad$Solve linear equation $\Gamma = \mathbf{E}\mathbf{S}' - \mathbf{E}'\mathbf{S} + \mathbf{E}'' + \Delta$ about $\mathbf{S}$;
9: $\quad$**return** $\mathbf{S}$;

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 can recover $\mathbf{S}$ by solving the linear equation $\Gamma = \mathbf{Q}$ if the attacker obtains Δ.

Algorithm 2 describes the KR-PCA on FrodoKEM. The attacker determines a reference plaintext and the corresponding valid reference ciphertext by performing an 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 $\Delta_{i,j}$) over the loop of Lines 3–7. At Line 6, we query 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 $\mathcal{O}((c_0, c_1^{(i,j,\delta-1)}), m) = 1$; in this manner, the attacker obtains information on $\Delta_{i,j}$ through the plaintext-checking oracle. Once 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 8.

In Lines 5–7, we require at most γ oracle accesses to determine $\Delta_{i,j}$ for each pair of i and j if using a naive manner. However, as Guo *et al.* mention in [GTN20], it is possible to 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, as mentioned in [BDL+19], the number of oracle accesses can be further reduced using a sparse ciphertext matrix. 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 obtain $\mathbf{M} = \mathbf{C} - \mathbf{D}^{(i)}\mathbf{S} = \mathbf{C} - \mathbf{Z}$, where the first row of $\mathbf{Z}$ is the i-th row of $\mathbf{S}$ and the remaining elements 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 the remaining rows are filled by 0's. We have $\mathbf{M}$ whose first row is $q/2^{B+1} - \mathbf{S}_{0,i}$ for $i = 0, 1, \ldots, \bar{n} - 1$, 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 fewer 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

The proposed SCA can be mounted on Kyber and Saber as there is similar KR-PCA using sparse ciphertexts and plaintext-checking oracle against them as that against FrodoKEM. Against Kyber, the proposed SCA can recover the secret key on the basis of key-recovery attack against Kyber-512 in Round 2 following the approach of Huguenin-Dumittan and Vaudenay [HV20] (precisely, we use the extended version in Xagawa *et al.* [XIU+21]). Against 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 will be 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

NTRU has two slightly differing KEM schemes—NTRU-HPS and NTRU-HRSS. In the PKE of both KEMs, the public key is h, the plaintext is a pair of "short" polynomials (r, m), and the 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 KR-PCA by Hoffstein and Silverman [HS99] and Jaulmes and Joux [JJ00] against the original NTRU, in which m is the plaintext and r is a randomness. These key-recovery attacks modify $c = h \cdot r + \mathsf{lift}(m)$ into $c' = c + \delta$ and check whether 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 remaining 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 primary hurdle to adapting this attack to NTRU is that NTRU's ciphertext space is changed from the original NTRU. Thus, δ should also satisfy $\delta \equiv 0 \pmod{(q, x-1)}$. This constraint makes analysis complex, and therefore, we do not adopt these attacks.

We can also use the KR-PCA attack against NTRU-HPS and NTRU-HRSS by [DDS+19] and [ZCQD21], respectively. These key-recovery attacks fix $m_{\mathsf{guess}} = 0$, modify r' and r_{guess}, compute $c = h \cdot r'$, and check whether the decrypted plaintext (r, m) is equivalent to the guess $(r_{\mathsf{guess}}, m_{\mathsf{guess}})$ or not. In both 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 Rounds 2 and 3 uses SXY [SXY18] as a variant of the FO transformation; this approach does not involve the computation of $r' \leftarrow G(m')$ because the underlying PKE.Enc is *deterministic*. Furthermore, NTRU does not perform the re-encryption test explicitly. However, NTRU still involves the validity check in the decapsulation, which can be exploited *via* the framework of the proposed SCA. In addition, (un)fortunately, NTRU's decapsulation program in `pqm4` computes both keys $k = \mathsf{H}(r, m)$ and $k' = \mathsf{H}_{\mathsf{prf}}(s, c)$ and outputs one of them according to the result of the implicit re-encryption test. In our experiments, we are able to detect whether $m_{\mathsf{guess}} = 0$ or not from the leakage of these computations of H and $\mathsf{H}_{\mathsf{prf}}$ or the procedure corresponding to the validity check with a high accuracy. (See Section 6 for the details.)

4.1.5 NTRU Prime

NTRU Prime has two KEM schemes: sntrupr (Streamlined NTRU Prime) and ntrulpr (NTRU LPRime). As NTRU LPRime has a similar structure to Kyber, Saber, and FrodoKEM, it is possible to mount a KR-PCA on it following the approach in [XIU+21], in which 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 structure similar to that of NTRU. The plaintext is r and a ciphertext is $c = \mathsf{Round}(h \cdot r)$, where $\mathsf{Round}(x)$ rounds each coefficient of x to

the nearest element in $3\mathbb{Z}$.[1] However, there are some technical hurdles to adapting conventional KR-PCAs (or KR-PCAs similar to the above) against NTRU. For example, Streamlined NTRU Prime's PKE.Dec internally checks the Hamming weight of each decrypted plaintext r and overwrites the decrypted plaintext with the fixed plaintext r_{fixed} if the test fails. Very recently, Ravi *et al.* [REB+21] propose two key-recovery side-channel attacks against Streamlined NTRU Prime, which is inspired by chosen-ciphertext attacks against NTRU proposed by Jaulmes and Joux [JJ00].

The first attack is based on the "plaintext-checking" oracle, which tests whether the internal variable is 0 or not. The internal decrypted plaintext will be either 0 or some polynomial; in both cases, the Hamming weight will be invalid and the output of the underlying PKE.Dec is r_{fixed}. Thus, in order to implement this "plaintext-checking" oracle, we need to analyze side-channel information of the computation in $\mathsf{PKE.Dec}(\mathsf{sk}, c)$ rather than PRF, which is out of focus of this paper.

The second attack is based on the decryption-failure oracle, which tests whether the decrypted plaintext is intended to be r or not. If so, the Hamming weight of r will be valid. In contrast, if the decryption failure occurs, the Hamming weight of the decrypted plaintext becomes invalid and it is overwritten by r_{fixed}. Ravi *et al.* implement the decryption-failure oracle by analyzing side-channel information from the re-encryption test. We can adopt and modify their attack for the proposed SCA, which indicates that the proposed general framework can be instantiated with a decryption failure oracle for the application to Streamlined NTRU Prime as follows:

Ravi *et al.*'s decryption failure-based attack and our modification: This attack proceeds in two phases:

1. In the first phase, the attack seeks a δ corresponding to a "1-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}. 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 of successfully obtaining 1-collision.[2] We then follow their strategy to design δ and estimate the probability of successfully obtaining appropriate δ as approximately 1% and 1.5% for sntrup653 and sntrup1277, respectively.[3] See [REB+21, Section 4.1 and 4.2] for the details.

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

We note that NTRU Prime uses a variant of the FO transformation that does not involve the computation of randomness because the underlying PKE.Enc is *deterministic* as NTRU. (Un)fortunately, NTRU Prime uses the explicit re-encryption test and also 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 desired.

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

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

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

4.2 Code-based KEMs

4.2.1 HQC

Roughly speaking, HQC has a structure similar to those of the lattice-based KEM schemes Kyber, Saber, FrodoKEM, and NTRU LPRime, even though HQC is based on the code problem. Hence, we can perform KR-PCAs on HQC in the strategy similar to them. Indeed, Huguenin-Dumittan and Vaudenay [HV20] give a KR-PCA against HQC in Round 2 by mimicking the attack by Băetu *et al.* [BDL+19] against another code-based PKE Lepton [YZ17]. Although HQC changed the parameters and decoder from Rounds 2 to Round 3, adjusting the parameter setting enables us to perform the KR-PCA; see Xagawa *et al.* [XIU+21] for details. 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 to obtain the decrypted plaintext in the re-encryption, the plaintext-checking oracle can be enacted using the SHAKE leakage *via* the proposed SCA.

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. In [GJS16], Guo *et al.* give a key-recovery reaction attack (GJS attack) against QC-MDPC [MTSB13], which is a variant of the McEliece PKE with QC-MDPC codes. Roughly speaking, the decryption oracle can be used to recover the distance profile $\mu(h_0)$ of one-half of a secret key $h_0 \in \mathrm{GF}(2)^n$. The distance profile contains (d, μ_d) for $d = 1, 2, \ldots, n/2$, implying that there are μ_d pairs of 1's with distance d in h_0. Guo *et al.* report that it is possible in practice to recover h_0 from its distance profile $\mu(h_0)$ in the parameter set for 80-bit security. Xagawa *et al.* [XIU+21] report that the GJS attack [GJS16] against QC-MDPC can be *partially* applied to BIKE in round 3 in the presence of the plaintext-checking oracle. Their approach recovers approximately one-quarter of the distance profile in the parameter set for 128-bit security. The decapsulation of BIKE employs 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 multiple ciphertexts (*e.g.*, 2,000 ciphertexts for each d) from crafted invalid plaintexts *at random* in order to compute (d, μ_d) and checks whether they are decrypted correctly or not; as a result, it is impossible to fix the template plaintext.

4.2.3 Classic McEliece

The proposed attack is not applicable to Classic McEliece because there is no known adaptive attack on the PKE of Classic McEliece. 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 KR-PCA is discovered.

4.3 Isogeny-based KEM

Hereafter, we propose a new SCA on SIKE focusing on the FO transformation. The proposed SCA is based on an adaptive attack on Jao's and De Fao's supersingular isogeny cryptosystem [JDF11] (namely, supersingular isogeny Diffie–Hellman (SIDH)) proposed by Galbraith *et al.* [GPST16]. We describe a modification of their attack for mounting the proposed SCA on SIKE.Decaps in the proposed framework.

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 over $\mathbb{F}_{p^2}$ with $p = 2^{e_A}3^{e_B} \pm 1$ (in SIKE in Round 3, E_0 is defined as

$y = x^3 + 6x^2 + x$). Let sk_2 and sk_3 be Alice's and Bob's secret keys, respectively. Let $R_A = P_A + [\mathsf{sk}_2]Q_A$ and $R_B = P_B + [\mathsf{sk}_3]Q_B$ be the secret points for generating finite cyclic groups for the kernels of Alice's and Bob's isogenies ϕ_A and ϕ_B, respectively. As well, let pk_2 and pk_3 be Alice's and Bob's public keys, respectively. 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). As sk_3 acts as the secret key in SIKE, the goal of an adaptive attacker is to recover sk_3 by adaptively querying ciphertexts to the decryption oracle. Note that, in SIKE, Alice corresponds to the sender (and attacker in the proposed SCA) and Bob corresponds to the receiver with a key generation (and victim).

At SIKE.Encaps, Alice computes her secret point R_A and public curve E_A (and $\tilde{P_A}$ and $\tilde{Q_A}$) to generate 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 $\langle R_A, R_B \rangle \coloneqq \{\, [n_A]R_A + [n_B]R_B \mid n_A \in \{0, 1, \ldots, 2^{e_A} - 1\}, n_B \in \{0, 1, \ldots, 3^{e_B} - 1\} \,\}$. The valid ciphertext of SIKE.Encaps is given by $c_0 = \mathsf{pk}_2 = (E_A, \tilde{P_A}, \tilde{Q_A})$ [4] 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, we consider a secret key with a ternary digit representation $\mathsf{sk}_3 = 3^0\beta_0 + 3^1\beta_1 + \cdots + 3^i\beta_i + \cdots + 3^{e_B-1}\beta_{e_B-1}$ where $\beta_i \in \{0, 1, 2\}$. The adaptive attack performs the key recovery from the least significant ternary digit upto the most significant ternary digit 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, that is, $\beta_0, \beta_1, \ldots, \beta_{i-1}$ (the attack description includes the initial case, namely $i = 0$). 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

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

respectively. 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). In other words, the decryption oracle correctly calculates the cyclic group generator as R_{AB} for a valid ciphertext. On the other hand, when querying $(c_0^{(\tau,i)}, c_1)$ to the decryption oracle, the generator of the cyclic group is calculated as

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

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

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

which follows from the fact that the order of $\tilde{Q_A}$ is 3^{e_B}. Thus, if $R_{AB}^{(\tau,i)} = R_{AB}$ (*i.e.*, $\tau = \beta_i$), the PKE decryption result is equivalent to the reference plaintext; otherwise, the

[4] In practice, instead of E_A, 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_A is reconstructed from the three points by the receiver. This has no influence on the adaptive attack.

Algorithm 3 Key-recovery plaintext-checking attack on SIKE

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

1: **Function** AttackOnSIKE$((c_0, c_1), m)$
2: $K_0 \leftarrow 0$;
3: **for** $i = 0$ to $e_B - 1$ **do**
4: **for each** $\tau \in \{0, 1, 2\}$ **do**
5: $\tilde{P}_A^{(\tau,i)} \leftarrow \tilde{P}_A - [3^{e_B-i-1}K_i + 3^{e_B-1}\tau]\tilde{Q}_A$;
6: $\tilde{Q}_A^{(\tau,i)} \leftarrow \tilde{Q}_A + [3^{e_B-i-1}]\tilde{Q}_A$;
7: $(c_0^{(\tau,i)}, c_1) \leftarrow ((E_A, \tilde{P}_A^{(\tau,i)}, \tilde{Q}_A^{(\tau,i)}), c_1)$;
8: **if** $\mathcal{O}((c_0^{(\tau,i)}, c_1), m) = 1$ **then**
9: $\beta_i \leftarrow \tau$;
10: $K_{i+1} \leftarrow K_i + 3^i\beta_i$;
11: **return** K_{e_B};

PKE decryption result is different from the reference plaintext. Therefore, the attacker can obtain the i-th ternary digit of the secret key (*i.e.*, β_i) *via* a plaintext-checking oracle. Thus, the attack recovers β_i in an iterative manner using a 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 KR-PCA on SIKE. In SIKE.Decaps, the j-variant value depends only on 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) to 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 needed to carry out 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 $\mathcal{O}((c_0^{(1,i)}, c_1), m) = 0$. We can also use the SHAKE leakage inside the PKE decryption (*i.e.*, $\mathsf{SHAKE}(j(E_0/\langle R_A, R_B\rangle))$, and it is 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 general 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 $\{0, 1, \ldots, \ell_B - 1\}$.

4.4 Complexity analysis

Table 3 lists the number of oracle accesses required for the proposed SCA to recover the full key of NIST PQC third-round candidates for KEM. For the simplicity, Table 3 lists only the results for instances with security levels equivalent to AES128 and AES256 (*i.e.*, NIST security levels 1 and 5, respectively).

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

Relative to previous CCA SCAs on lattice-based KEMs (*e.g.*, [XPRO20, RBRC20, SKL+20, NDGJ21]), the proposed attack may require more oracle accesses. This would be because some previous SCAs exploit scheme/implementation-specific aspects for improved efficiency in terms of the number of traces, although the proposed SCA is applicable to (relatively) black-box implementation. Furthermore, the attack by Ravi *et*

Table 3: Number of oracle accesses required by proposed attack for key recovery (except for 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	ntruhrss701	≈ 2804 ($= 4 \times 701$)
		ntruhps2048509	≈ 1018 ($= 2 \times 509$)
		ntruhps4096821	≈ 1642 ($= 2 \times 821$)
	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. ($= 100/1.5 + 4 \times 1277$)
Code	HQC	hqc128	≈ 18111 ($= 46 + \log(46) + 46 \times (384 + \log(384))$)
		hqc256	≈ 58536 ($= 90 + \log(90) + 90 \times (640 + \log(640))$)
	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 1,500 distance profiles out of 6,162 full-distance profiles for Level 1. It is difficult to reliably recover key bits more than this using the proposed SCA.

al. [RRCB20] can also perform full-key recoveries of Kyber, Saber, and FrodoKEM with a complexity comparable to the proposed SCA. We stress that the primary advantage of the proposed attack is the generality, as the proposed SCA is applicable to many lattice-, code-, and isogeny-based KEMs. Owing to the high applicability of adaptive attack using the plaintext-checking oracle, the proposed SCA offers a higher degree of generality for KEMs based on the FO transformation and its variants.

5 Side-Channel Distinguisher Design

This section describes the design of a DL-based side-channel distinguisher. In recent years, several studies have evaluated and demonstrated the significant advantage of DL in carrying out profiling SCAs (*e.g.*, [BPS+18, KPH+19, PCP20, ZBHV20, WAGP20]). In a DL-based profiling SCA, a trained NN is used to estimate intermediate value (or its Hamming weight/distance) from side-channel leakage, and the secret key is estimated using the likelihood from the NN output (*i.e.*, occurrence probability for intermediate value or its Hamming weight/distance). Therefore, the conventional DL-based SCA on AES usually utilizes an NN with nine 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 fewer traces (*e.g.*, [ZBHV20, WAGP20]). In the following experiment, we employ a convolutional NN (CNN) and a multilayer perceptron (MLP), as the practicality and effectiveness of CNN and MLP are shown in several previous studies on DL-based SCA. The CNN/MLP for our experiment is designed to have a sufficient model capacity for application to various PRF implementations including software and hardware with and without masking countermeasure. The proposed SCA should perform two-classification of whether the input to the PRF is the reference plaintext or not; therefore, we construct a CNN/MLP 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 use a standard CNN/MLP 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 oracle, because an error in oracle would render the recovered key critically incorrect. However, the accuracy of an NN model used for SCA is occasionally nonnegligibly low owing to the presence of noise and SCA countermeasure [ISUH21]. To improve the oracle accuracy 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 PRF input 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 the overall 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 3.

However, such a majority vote considers the NN output to be a binary value, although the NN output is given as a probability of the PRF input being the reference plaintext; this suggests that the majority vote does not fully exploit the advantages of the NN-based distinguisher. As an efficient alternative, we can determine the plaintext-checking oracle output by means of the likelihood comparison, in which 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. Because such a method exploits the NN output more effectively than the above majority vote, it can enact the plaintext-checking oracle more accurately. Actually, if $q(b \mid \boldsymbol{x}_s; \hat{\theta})$ has been sufficiently trained and approximates the true distribution, such a likelihood ratio test (herein, NLL comparison) 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 its effectiveness and practicality.

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 to carry out the NN training. The learning rate was 0.001, the batch size was 32, and the number of epochs was 100. Table 4 summarizes the hyper parameters of the CNN for traces with 1,000 sample points, where the top and bottom columns denote the input and output layers, respectively, and the remaining hidden layers are connected in the ascending order from the input to the output. In the "Input" row, $S_1 \times S_2$ denotes the input shape, S_1 is the traces size, and S_2 is the input dimension. In the "Operator" row, conv1d(F) denotes the operation at the each layer and F is the filter size. (a) For non-protected software,

Table 4: NN hyper parameters

(a) For non-protected software, non-protected hardware, and masked hardware implementations

	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	125×4	conv1d(3)	8	SELU	Yes	Avg (2)	2
Conv5	62×8	conv1d(3)	8	SELU	Yes	Avg (2)	2
Conv6	31×8	conv1d(3)	8	SELU	Yes	Avg (2)	2
FLT	15×8	flatten	120	-	-	-	-
FC1	120	dense	20	SELU	No	No	-
FC2	20	dense	20	SELU	No	No	-
FC3	20	dense	1	Sigmoid	No	No	-

(b) For masked software implementation

	Input	Operator	Output	Activation function	Batch normalization	Pooling	Stride
FC1	100	dense	32	SELU	No	No	-
FC2	32	dense	32	SELU	No	No	-
FC3	32	dense	20	SELU	No	No	-
FC4	20	dense	20	SELU	No	No	-
FC5	20	dense	1	Sigmoid	No	No	-

non-protected hardware, and masked hardware implementations, we employ a CNN comprising six convolutional layers *Conv1*, *Conv2*, ..., and *Conv6* followed by three fully connected layers *FC1*, *FC2*, and *FC3*, as the effectiveness of CNN in DL-based SCA has been shown in many previous studies (*e.g.*, [BPS+18, KPH+19, PCP20, ZBHV20, WAGP20]). (b) For masked software implementation, we employ an MLP which consists of five fully connected layers *FC1*, *FC2*, ..., and *FC5*, as we expect that a cascade of fully connected layers may exploit multivariate leakages of masked software more efficiently than a CNN. The output layer (*i.e.*, (a) *FC3* and (b) *FC5*) has one output for two-classification. Given a side-channel trace, the CNN/MLP outputs the probability that the plaintext (*i.e.*, PRF input) is equal to the reference plaintext (or, conversely, other plaintext).

Table 5 lists the experimental implementations and the numbers of traces used for the experiments.[5] We employ the following five PRF implementations: non-protected AES and SHAKE softwares, non-protected AES hardware, masked AES software, and masked AES hardware. For masked implementations, we target a bit-sliced software and TI-based hardware, which are one of the most promising first-order masking schemes for software and hardware implementations, respectively. We use major publicly-available open-source implementations of non-protected software/hardware and masked software for the reproducibility. In contrast, the masked hardware is implemented by ourselves according to the paper [UHA17], because there are no publicly-available TI-based masked hardware. The source code of the masked hardware is included in our repository. Although SHAKE is more commonly used than AES as PRF/PRG in KEMs, we 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 results produced by AES and SHAKE, if they are protected using the same masking scheme.

[5] 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 the validity check in `owcpa_dec`, instead of a PRF in NTRU.Decaps, whereas the key recovery of NTRU can be also performed using the leakage of KDF as described in Section 4.1.4. As a result, we confirmed that the DL-based distinguisher achieves a 99.8% accuracy, and its combination with likelihood comparison achieves a 100% test accuracy with only two traces.

Table 5: Experimental implementations and numbers of used traces

	Non-protected AES/SHAKE software	Non-protected AES hardware	Masked bit-sliced AES software	Masked AES hardware based on TI
Reference	`pqm4` [KRSS19, pqm21]	SASEBO IP [Toh]	Schwabe and Stoffelen [SS16, git21]	Ueno *et al.* [UHA17]
Device	STM32F415RGT6	Xilinx Kintex-7	STM32F407VGT6U	Xilinx Kintex-7
Board	NewAE Technology STM32F	SAKURA-X	STM32F407G-DISC1	SAKURA-X
Side-channel trace	Supply voltage current	Supply voltage current	EM radiation	Supply voltage current
Measurement interface	NewAE technology chip-whisperer CW308	On-board coaxial connector	Langer EMV-Technik RF-U T-2 probe	On-board coaxial connector
Oscilloscope	Keysight Technologies MSOX6004A			
# Training traces	30,000	30,000	900,000	980,000
# Validation traces	10,000			
# Test traces	10,000			

To evaluate performance of the distinguish attack (*i.e.*, the model accuracy for enacting the plaintext-checking oracle), the side-channel traces are given in a manner similar to the fixed-vs.-random TVLA as follows: For AES, the plaintext is given by a fixed or random value to be distinguished, and the key is fixed for both fixed and random plaintexts.[6] For SHAKE, the input is given in the same manner as the plaintext for the AES case.

For masked implementations, we explicitly use traces during only masked operations, and we excluded timings corresponding to initial masking. As there are some end-to-end masked implementations that masks all decapsulation procedures including PKE.Dec, re-encryption, and equality check [OSPG18, BPO+20], we exploit the leakage from only masked operations in attacking such an implementation. More precisely, for masked software implementation, we set a trigger at the timing just before the start of first round after the initial masking, and use traces during masked AES operations. For masked hardware implementation, we acquire the traces during the masked first-round computation.

6.2 Accuracy evaluation

Table 6 reports the accuracy of the trained NN on the test sets. From Table 6, we can confirm that the trained NN achieved a meaningfully high accuracy to carry out the proposed SCA except for masked hardware. For the non-protected software and hardware implementations, the NN model achieves a 99.8% and 99.9% test accuracy, respectively. In addition, the NN model achieves a 96.0% accuracy even for the masked software. The masking countermeasure reduces the performance of the distinguish attack. However, for software implementation, the mitigation would not be sufficient to prevent the attack using multiple traces. In contrast, it is difficult to distinguish the input of masked hardware due to the advantage of TI.

[6] In using AES as an XOF for modern KEMs (*e.g.*, Kyber, NTRU LPRime, and BIKE), AES frequently works in the CTR mode in which the plaintext is frequently fixed and the key is the payload. However, we set the plaintext as the payload and set the key fixed in the experiment. This is because we 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 implementations (*e.g.*, masked AES software in [SS16] and hardware in [UHA17], which are used in our experiment) do not protect the key scheduling parts because it causes no DPA leakage, although it causes an exploitable leakage for the distinguish attack. Therefore, for a general and severe evaluation, 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. (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 even for the distinguish attack, because the key scheduling part always processes an identical value for a fixed key. By contrast, if the key is a seed and the plaintext is fixed, both round datapath and key scheduling datapath are leakage sources for the distinguish attack. Thus, the experiment validates the proposed attack in a more general and severe

Table 6: Accuracy of NN to distinguish PRF input

	Non-protected software	Non-protected hardware	Masked software	Masked hardware
Accuracy	0.998	0.999	0.960	0.515

We then evaluate the oracle accuracy in using the majority vote and likelihood comparison. The majority vote is evaluated in an analytical manner using Eq. (1) for odd numbers of traces, whereas the likelihood comparison is evaluated experimentally using a shuffled test data repeatedly. More precisely, to evaluate the accuracy using the likelihood comparison using t traces, we repeat the following procedure 10,000 times: we randomly determine $b_{\text{true}} = 0$ or 1, obtain t traces for reference plaintext from the test set if $b_{\text{true}} = 1$; otherwise (*i.e.*, if $b_{\text{true}} = 0$) for random plaintext, calculate the NLL in Eq. (2) for each hypothetical oracle output $b = 0$ or 1, determine the oracle output b as the smaller NLL, and examine whether $b_{\text{true}} = b$. The distinguisher with a likelihood comparison can achieve a 100.0% test accuracy with at least 2, 2, and 5 traces for non-protected software, non-protected hardware, and masked software, respectively. In contrast, the majority vote requires 5, 5, and 11 traces for a 99.999% accuracy[7] for non-protected software, non-protected hardware, and masked software, respectively. Thus, we can confirm that the likelihood comparison would be more accurate and effective than majority vote according to the Neyman–Peason lemma, and the trained NNs can achieve a sufficient accuracy for the key recovery except for masked hardware. In contrast, we find that we cannot achieve a high accuracy (more than 99.999%) fewer than 5,000 traces in our environment, which indicates that the proposed attack cannot break masked TI-based hardware using our NN model.

6.3 Evaluation of number of traces for successful key recovery

Table 7 lists the number of side-channel traces required for a successful key recovery when using the side-channel distinguisher evaluated in the previous subsection (except for masked hardware). For non-protected implementations and masked software, we adopt the distinguisher based on the likelihood comparison with 2 and 5 traces according to the evaluation in Section 6.2. The results listed in Table 7 assume that the oracle enacted by the side-channel distinguisher is completely accurate if it achieves a 100.0% test accuracy.

If a larger number of traces is needed to enact an accurate oracle, more traces are obviously required for key recovery. However, our experimental results reveal that the attack is still feasible on KEMs even if the PRF implementation is masked in software, given that the modern SCA evaluation is conducted with more-than 10M or 100M traces (*e.g.*, [SM15,SM19,SBM19]). At least, the first-order masking countermeasures on software are not an essential solution to counter the proposed SCA. Evaluation of the DL-based distinguish attack on higher-order masked implementation will be the subject of important future work.

In contrast, Table 7 does not include the results on TI-based masked hardware, as we could not achieve a sufficient accuracy for mounting the KR-PCA. Thus, TI would be an effective countermeasure against the proposed attack. However, even TI may be broken owing to the development of DL-based SCAs on masked hardware. In fact, most existing studies on DL-based SCA focus on masked software implementation, whereas very few literature demonstrates attack on masked hardware. The investigation of DL-based SCAs on masked hardware is also an important future work.

manner, as the experiment is harder for the attacker than the practice.)

[7] Note that the majority vote cannot achieve a 100% accuracy from a model with less than 100.0% test accuracy as in Eq. (1).

Table 7: Number of side-channel traces required for successful proposed attack (partial-key recovery for BIKE and except for Classic McEliece)

KEM type	Scheme	Instance	# Traces for attack phase	
			Non-masked implementations	Masked software
Lattice	Kyber	Kyber-512	3,072	7,680
		Kyber-1024	6,144	15,360
	Saber	LightSaber-KEM	6,144	15,360
		FireSaber-KEM	6,144	15,360
	FrodoKEM	FrodoKEM-640	51,200	128,000
		FrodoKEM-1344	86,016	215,040
	NTRU	ntruhrss701	5,608	14,020
		ntruhps2048509	2,036	5,090
		ntruhps4096821	3,284	8,210
	NTRU Prime	ntrulpr653	2,612	6,530
		ntrulpr1277	5,108	12,770
		sntrup653	5,424	13,560
		sntrup1277	10,350	25,875
Code	HQC	hqc128	36,222	90,555
		hqc256	117,072	292,680
	BIKE	Level 1	6M	15M
		Level 5	N/A	N/A
	Classic McEliece	Any	N/A	N/A
Isogeny	SIKE	SIKEp434	548	1,370
		SIKEp751	956	2,390

7 Conclusion

This paper presented a generic power/EM attack methodology targeting KEMs based on the FO transformation and its variant using a plaintext-checking oracle. 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 PRF input is equal to the reference plaintext or not. We demonstrated that all KEMs in the NIST PQC third-round candidates except for Classic McEliece are vulnerable to the proposed attack. We also presented a DL-based side-channel distinguisher design, which was demonstrated through experimental attacks on various PRF implementations, including implementations protected by a masking countermeasure. Our results confirm that the proposed SCA can perform key recoveries on many KEM implementations, even if the PRF implementation is protected in software. Meanwhile, we also confirm that the proposed attack was not successful on masked TI-based hardware in our environment, which would be an effective countermeasure against the proposed attack as well as existing key-recovery SCAs.

The proposed attack has two significant advantages: the generality and applicability. First, the proposed attack realizes a plaintext-checking oracle through a side-channel leakage of PRF in the re-encryption. Since many PKEs 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. In addition, the PRF and equality/validity check play an essential role in KEMs with a CPA-to-CCA-secure transformation. Although some CPA-to-CCA-secure transforms do not perform a complete re-encryption, the proposed SCA would be applicable even to such variants of FO transformation as long as they employ PRF and/or procedure corresponding to the validity check. Second, the proposed SCA does not require the detailed knowledge of target implementation and therefore can be applied to (relatively) black-box implementations. Thus, in 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 secret key, suggesting that KEM implementations should be resistant to profiling attacks including DL-based ones to counter the proposed SCA. Evaluation of higher-order masking against the proposed attack and developing an effective countermeasure will be the important future work for realizing a secure KEM implementation. As well, the capability evaluation of DL-based SCA on masked hardware is an important subject to validate the security of masked TI-based hardware against the proposed attack. We are also planning to investigate the applicability of the proposed SCA to KEMs others than the NIST PQC third-round candidates.

Acknowledgment

We would like to thank Prof. Jean-Sébasiten Coron for his shepherding care. This work has been supported by JSPS Kakanhi Grant No. 17H00729 and 19H21526, JST CREST No. JPMJCR19K5, and JST PRESTO No. JPMJPR18M3.

References

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

[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*, 2021:334–359, 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.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology—CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Seience*, pages 1–12, 1998.

[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.

[DOV21] Jan-Pieter D'Anvers, Emmanuela Orsini, 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.

[ISUH21] Akira Ito, Kotaro Saito, Rei Ueno, and Naofumi Homma. Imbalanced data problems in deep learning-based side-channel attacks: Analysis and solution. *IEEE Transactions on Forensics and Security*, 2021. DOI: 10.1109/TIFS.2021.3092050.

[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.

[MWM21] Thorben Moos, Ferix Wegener, and Amir Moradi. DL-LA: Deep learning leakage assessment—a modern roadmap for SCA evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 552–598, 2021.

[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.

[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.

[Toh] Tohoku University. Cryptographic hardware project. http://www.aoki.ecei.tohoku.ac.jp/crypto/.

[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.