Fault-Injection Attacks against NIST's Post-Quantum Cryptography Round 3 KEM Candidates

Keita Xagawa[3], Akira Ito[1], Rei Ueno[12], Junko Takahashi[3], and Naofumi Homma[1]

[1] Tohoku University
2–1–1 Katahira, Aoba-ku, Sendai-shi, 980-8577, Japan
ito@riec.tohoku.ac.jp,rei.ueno.a8@tohoku.ac.jp,homma@riec.tohoku.ac.jp
[2] PRESTO, Japan Science and Technology Agency
4–1–8 Honcho, Kawaguchi, Saitama, 332-0012, Japan
[3] NTT Secure Platform Laboratories
3-9-11, Midori-cho Musashino-shi, Tokyo 180-8585 Japan
junko.takahashi.fc@hco.ntt.co.jp,keita.xagawa.zv@hco.ntt.co.jp

Abstract. We investigate *all* NIST PQC Round 3 KEM candidates from the viewpoint of fault-injection attacks: Classic McEliece, Kyber, NTRU, Saber, BIKE, FrodoKEM, HQC, NTRU Prime, and SIKE. All KEM schemes use variants of the Fujisaki-Okamoto transformation, so the equality test of re-encryption in decapsulation is critical.
We survey effective key-recovery attacks if we can skip the equality test. We found the existing key-recovery attacks against Kyber, NTRU, Saber, FrodoKEM, HQC, one of two KEM schemes in NTRU Prime, and SIKE. We propose a new key-recovery attack against the other KEM scheme in NTRU Prime. We also report an attack against BIKE that leads to leakage of information of secret keys.
The open-source pqm4 library contains all KEM schemes except Classic McEliece and HQC. We show that giving a single instruction-skipping fault in the decapsulation processes leads to skipping the equality test *virtually* for Kyber, NTRU, Saber, BIKE, and SIKE. We also report the experimental attacks against them. We also report the implementation of NTRU Prime allows chosen-ciphertext attacks freely and the timing side-channel of FrodoKEM reported in Guo, Johansson, and Nilsson (CRYPTO 2020) remains.
keywords: post-quantum cryptography, NIST PQC standardization, KEM, the Fujisaki-Okamoto transformation, fault-injection attacks.

1 Introduction

Key encapsulation mechanism: Key encapsulation mechanism (KEM in short) [Sho00, CS03, ISO06] is a fundamental cryptographic primitive, which can be considered as a variant of public-key encryption (PKE). KEM's encryption algorithm, which we call the encapsulation algorithm, takes a public key as input and outputs a ciphertext and a key. KEM's decryption algorithm, which we call the decapsulation algorithm, takes a secret key and a ciphertext as input and outputs a key instead of a message. KEM is a versatile primitive and has a lot of applications, e.g., key exchange, hybrid encryption, secure authentication, and authenticated key exchange.

The most standard security notion of KEM is indistinguishability against chosen-ciphertext attacks (IND-CCA-security) [RS92, CS03]. Since it is hard to construct efficient IND-CCA-secure KEMs directly, cryptographers often use the transformations from weakly-secure PKE/KEM into IND-CCA-secure KEM. The Fujisaki-Okamoto (FO) transformation [FO99, FO13, Den03] is one of the transformations often used in the design of IND-CCA-secure PKE/KEM in the random oracle model (ROM). Roughly speaking, the FO transformation for KEM transforms an underlying PKE scheme into KEM as follows: A key-generation algorithm of KEM is the same as that of PKE. An encapsulation algorithm on input pk chooses a message m randomly, encrypts it into $ct = \mathsf{Enc}(pk, m; \mathsf{G}(m))$, where the randomness of encryption is computed as $\mathsf{G}(m)$, and outputs a ciphertext ct and a key $K = \mathsf{KDF}(m)$. A decapsulation algorithm on input sk and ct decrypts ct into $m' = \mathsf{Dec}(sk, ct)$, re-encrypts m' into $ct' = \mathsf{Enc}(pk, m'; \mathsf{G}(m'))$, and outputs a key $K = \mathsf{KDF}(m')$ if $ct = ct'$ and a rejection symbol otherwise.

Post-quantum cryptography: Scalable quantum computers will threaten classical public-key cryptography since Shor's algorithm on a quantum machine solves factorization and discrete logarithms efficiently [Sho94]. The recent progress in developing quantum machines motivates us to replace classical public-key cryptography with post-quantum cryptography (PQC). Hence, in the past decade, the security proofs of the FO transformation and its variants have been extended to those in the quantum random oracle model (QROM) [BDF+11] to show the security against *quantum* polynomial-time adversary. See e.g., [TU16, HHK17, SXY18, JZC+18, BHH+19, JZM19, KSS+20].

Moreover, in 2016, NIST PQC standardization called for proposals on PKE/KEM and signatures as the basic primitives [4]. In 2020, NIST select four finalists and five alternate candidates for KEM in Round 3 [AAA+20]. All use the FO-like transformations to construct IND-CCA-secure KEMs in the (Q)ROM.

Fault-injection attacks: In the real world, the decapsulation algorithm is implemented physically. Hence, investigations into side-channel attacks (SCA) [Koc96] and fault-injection attacks (FIA) [BDL01] against proposed KEMs are strongly promoted by NIST.

We focus on FIA against KEM and review the scenario of it. Suppose that an adversary can inject faults into a decapsulation machine that contains a secret key. In this situation, it is natural to think the adversary has the machine itself and uses it freely because the adversary can physically access the machine. Hence, the adversary can decrypt any ciphertexts and recover corresponding messages. If we consider FIA, message-recovery attacks are not so important.

On the other hand, key recovery via FIA is non-trivial and interesting, because the key-recovery attack logically breaks a tamper-resilient memory by extracting the secret from it. In addition, once one obtains a secret key of a decapsulation machine, one can copy the machine. Thus, we examine how FIA leads to a key-recovery attack.

Skipping-the-equality-test attack: In the FO-like transformations, the decapsulation algorithm given a ciphertext ct first decrypts the ciphertext into m', re-encrypts it into ct', and returns $K = \mathsf{KDF}(m', \mathsf{aux})$ if $ct = ct'$ and pseudorandom value $K = \mathsf{KDF}(s, \mathsf{aux})$ or the rejection symbol $\perp$ otherwise, where aux depends on pk and ct and s is a secret value.

By injecting a fault carefully, we could force the decapsulation machine to *skip* the equality test $ct = ct'$ and return $K = \mathsf{KDF}(m', \mathsf{aux})$ always, where $m' = \mathsf{Dec}(sk, ct)$. This enables us to implement a plaintext-checking oracle on input guess m_{guess} and ciphertext ct by checking if $K = \mathsf{KDF}(m_{\mathrm{guess}}, \mathsf{aux})$ or not and a key-mismatch oracle on input guess K_{guess} and ciphertext ct by checking if $K = K_{\mathrm{guess}}$ or not. Such oracles would enable an adversary to mount a key-recovery attack against KEM.

Fault-injection attack against pre-quantum KEMs: Factoring/RSA-based PKE/KEM is vulnerable against FIA. For example, safe-error attacks [YJ00, YKLM02] are effective critical to guess a bit of secret key. They are also applicable to Discrete-logaritm (DL)-based PKE/KEM. DL-based PKE/KEM has several attack surfaces vulnerable to FIA. See, for example, invalid point/curve attacks [BMM00, BG15, ABM+03, TT19].

However, we note that the existing key-recovery FIAs do not target the equality test of the FO transformation. We could break IND-CCA security of such PKE/KEM by using malleability of the underlying PKE and skipping the equality test and implementing plaintext-checking oracle of the underlying PKE, which enables an adversary to check if the decryption of a queried ciphertext is equal to a queried plaintext or not. However, it is not known whether this plaintext-checking oracle (or even decryption oracle) enables us to recover the secret key of the underlying PKE, say, the textbook RSA. (See e.g., [BV98] and [AM09].)

Fault-injection attack against post-quantum KEMs: This situation is changed in post-quantum KEMs. Unfortunately, underlying PKEs in the post-quantum PKE/KEMs are often vulnerable to key-recovery chosen-ciphertext attacks. For example, Hall, Goldberg, and Schneier [HGS99] pointed out message-recovery and key-recovery chosen-ciphertext attacks against the McEliece PKE [McE78, Nie86] and the Ajtai-Dwork PKE [AD97], respectively. Fluhrer pointed out that a simple key-exchange scheme based on ring learning with errors (RLWE) is vulnerable to the key-mismatch attack if a user fixes its secret [Flu16]. Galbraith, Petit, Shani, and Ti [GPST16] gave a key-recovery key-mismatch attack against SIDH [JD11, DJP14] with fixed secret. Therefore, the equality test is an important target of FIA.

Although Pessl and Prokop [PP21] pointed out that the equality test is '*an obvious faulting target*,' we do not know how easily we can mount a skipping-the-equality-test attack by injecting a single fault against the implementations in the wild and how effective the skipping-the-equality-test attack is against the NIST PQC Round 3 KEM candidates.

[4] https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals

Table 1: Summary of our findings on NIST PQC Round 3 KEM Candidates (finalists and alternatives) and their implementations in pqm4: PCA implies plaintext-checking attack.

Name	Effect of PCA	Attack Surface in pqm4	Effect of FIA in pqm4
Classic McEliece [ABC+20]	Unknown	–	–
Kyber [SAB+20]	Key recovery	Skip	Key recovery
NTRU – ntruhps [CDH+20]	Key recovery	Skip	Key recovery
NTRU – ntruhrss [CDH+20]	Key recovery	Skip	Key recovery
Saber [DKR+20]	Key recovery	Skip	Key recovery
BIKE [ABB+20]	Key leakage (New)	Skip	Key leakage
FrodoKEM [NAB+20]	Key recovery	Timing bug	Key recovery
HQC [AAB+20]	Key recovery	–	–
NTRU Prime – sntrupr [BBC+20]	Key recovery	CCA bug	Key recovery
NTRU Prime – ntrulpr [BBC+20]	Key recovery (New)	CCA bug	Key recovery
SIKE [JAC+20]	Key recovery	Skip	Key recovery

1.1 Our Contribution

We systematically study how effective fault-injection attacks that lead to the skip of the equality test of FO-like transformations are against *all* KEMs in the NIST PQC Round 3 finalists and the alternatives: Classic McEliece [ABC+20], Kyber [SAB+20], NTRU (ntruhps and ntruhrss) [CDH+20], Saber [DKR+20], BIKE [ABB+20], FrodoKEM [NAB+20], HQC [AAB+20], NTRU Prime (sntrupr and ntrulpr) [BBC+20], and SIKE [JAC+20]. We summarize our findings in Table 1.

Theoretical analysis: We study whether the underlying PKEs of KEMs are resilient to key-recovery plaintext-checking attacks (KR-PCA) or not, since skipping the equality test enables an adversary to obtain $K = \mathsf{KDF}(\mathsf{Dec}(sk, ct), \mathsf{aux})$ instead of pseudorandom string or $\perp$ and to implement a plaintext-checking oracle easily.

We found that almost all PKEs except the underlying PKE of Classic McEliece leaks information of the decryption key in the presence of plaintext-checking oracle *in vitro*. Our findings are summarized as follows (see also Table 2):

Kyber, NTRU, Saber, FrodoKEM, HQC, sntrupr of NTRU Prime, and SIKE: We survey the literature and found that there are KR-PCAs against the underlying PKEs of Kyber, ntruhps and ntruhrss of NTRU, Saber, FrodoKEM, HQC, sntrupr of NTRU Prime, and SIKE.

ntrulpr of NTRU Prime: We propose a KR-PCA against the underlying PKE of NTRU LPRime (ntrulpr of NTRU Prime) by mimicking the KR-PCAs against the underlying PKEs of Saber and Kyber [HV20]. See section 4.

BIKE: The underlying PKE of BIKE in round 3 also leaks the secret key's information from the plaintext-checking oracle as QC-MDPC [MTSB13] is vulnerable to the KR-PCA proposed by Guo, Johansson, and Stankvoski [GJS16]. However, the change of a decoder algorithm in round 3 makes key-recovery attacks difficult. See subsection C.5.

Classic McEliece: There are no known KR-PCAs against the underlying PKE of Classic McEliece if the decoder in a decryption algorithm rejects invalid plaintexts [5] (We note that the specifications seem to allow the use of any decoder that decodes binary Goppa codes.)

Trade-off: Skipping the equality test enables the adversary to obtain $K = \mathsf{KDF}(m, \mathsf{aux})$ with $m = \mathsf{Dec}(sk, ct)$ rather than the plaintext-checking oracle. Thus, the adversary can check if $m = m_{\mathrm{guess}}$ by checking $K = \mathsf{KDF}(m_{\mathrm{guess}}, \mathsf{aux})$ from one query. If the number of candidates of m is small, then we can determine the value of m by an exhaustive search. By using this property, there are trade-offs between the computational cost and the number of queries in the cases of Kyber, Saber, FrodoKEM, and ntrulpr of NTRU Prime. See the details in section 4 for the case of ntrulpr of NTRU Prime.

Investigation of KEMs in pqm4: We investigate implementation of KEMs in pqm4 [KRSS], which include Kyber, NTRU (ntruhps and ntruhrss), Saber, BIKE, FrodoKEM, NTRUPrime (sntrupr and ntrulpr), and SIKE [6]

[5] The plaintext space is a set of n-dimensional vectors whose Hamming weight is t.

[6] We use 2021 Mar. 8 version. https://github.com/mupq/pqm4/commit/17e43e52e75ca5197c397362cab6bcf885712a71.

Table 2: Theoretical plaintext-checking attacks and key-mismatch attacks against the underlying PKEs of NIST PQC Round 3 KEM Candidates.

Name	Results
Classic McEliece [ABC+20]	Unknown
Kyber [SAB+20]	Key recovery [QCD19, RRCB20, HV20, QCZ+21]
NTRU – ntruhps [CDH+20]	Key recovery [DDS+19]
NTRU – ntruhrss [CDH+20]	Key recovery [ZCQD21]
Saber [DKR+20]	Key recovery [HV20, QCZ+21]
BIKE [ABB+20]	Key leakage (New, adapted KR-PCA against QC-MDPC [GJS16])
FrodoKEM [NAB+20]	Key recovery [BDH+19, RRCB20, VV20, QCZ+21]
HQC [AAB+20]	Key recovery [HV20]
NTRU Prime – sntrupr [BBC+20]	Key recovery [REB+21]
NTRU Prime – ntrulpr [BBC+20]	Key recovery (New, adapted KR-PCA against Kyber, Saber, and FrodoKEM)
SIKE [JAC+20]	Key recovery [GPST16]

NTRU Prime: In the implementation of NTRU Prime (sntrupr and ntrulpr), a decapsulation program contains a fatal bug that forces the result of the equality test to be true. [7] Thus, we can mount a chosen-ciphertext attack against them. See subsection 5.1.

FrodoKEM: In 2020, Guo et al. [GJN20] pointed out that the implementation of FrodoKEM (and HQC) contains a leaky equality test that leaks information of the secret key from the timing side channel and succeeded in mounting a key-recovery attack using the timing information. Although FrodoKEM in Round 3 repaired this leaky equality test, the bug still remains in the implementation in pqm4.[8] See subsection 5.2.

Kyber, NTRU, and Saber: They shared a same structure to compute a key. Roughly speaking, decapsulation programs use a flag for the equality test and overwrite the decrypted result m' by a secret seed s if the flag is set. This overwriting is done by a single function call of 'cmov' (conditional-move). (Un)fortunately, we can skip this function call by a single fault and mount FIA. See subsection 5.3

BIKE: The decapsulation program of BIKE in pqm4 computes mask, which is -1 or 0 depending on the re-encryption check, and overwrites the decryption result m' by a secret seed s as $m' \leftarrow (m' \wedge \neg\text{mask}) \vee (s \wedge \text{mask})$.[9] (Un)fortunately, we identify a single operation such that if we skip the operation, then mask is set to 0 always. Thus, we can skip the overwrite procedure virtually by a single fault. See subsection 5.4.

SIKE: The implementation of SIKE in pqm4 simply uses an 'if' statement to overwrite the decrypted result m' by a secret seed s. In the assembly code, this if-then-overwrite is implemented as 'compare' and 'conditional jump'. (Un)fortunately, we can skip this 'conditional jump' by a single fault. See subsection 5.5.

Experimental results: On the basis of our findings, we conduct the experimental skip attacks on Kyber, NTRU, Saber, BIKE, and SIKE. The target is STM32F415 whose core is ARM Cortex-M4, which is a de-facto standard platform as NIST suggested. We run 100 fault injections to each scheme and succeeded in injecting faults correctly with 15–50%.

1.2 Related Works

For PQC candidates and their implementation, we recommend the reader to read a survey written by Howe, Prest, and Apon [HPA21]. Ravi and Roy gave a lecture on SCAs and FIAs against lattice-based PQC candidates [RR21]. Costello wrote a survey of isogeny-based cryptography [Cos21]. Fot SCA and FIA against NIST PQC KEM Candidates, see our survey in Appendix C.

1.3 Organization

Section 2 reviews basic notions and notations. Section 3 reviews the variants of the FO transformation. Section 4 gives a key-recovery attack against ntrulpr of NTRU Prime using plaintext-checking oracle and discusses a trade-off between efficiency and the number of queries if we consider the fault-injection attack. Section 5 describes

[7] We report it in https://github.com/mupq/pqm4/issues/195

[8] pqm4 noticed this issue. See https://github.com/mupq/pqm4/issues/161.

[9] If mask $= 0$, then we have $m' \leftarrow m'$. Otherwise, we have $m' \leftarrow s$.

the equality test of KEMs and how we can mount skipping attack. Section 6 reports our experimental results. Appendix A contains missing definitions, say, security notions of PKE and KEM. Appendix B also reviews the variants of the FO transformation. Appendix C reviews the KEM schemes and KR-PCAs against of them and includes our report of key-leakage PCAs against BIKE and stnrupr of NTRU Prime.

2 Preliminaries

2.1 Notation

A security parameter is denoted by λ. We use the standard O-notations. DPT, PPT, and QPT stand for deterministic polynomial-time, probabilistic polynomial-time, and quantum polynomial-time, respectively. A function $f(\lambda)$ is said to be *negligible* if $f(\lambda) = \lambda^{-\omega(1)}$. We denote a set of negligible functions by $\mathsf{negl}(\lambda)$. For a statement P (e.g., $r \in [0, 1]$), we define $\mathsf{boole}(P) = 1$ if P is satisfied and 0 otherwise.

For a distribution χ, we often write "$x \leftarrow \chi$," which indicates that we take a sample x in accordance with χ. For a finite set S, $U(S)$ denotes the uniform distribution over S. We often write "$x \leftarrow S$" instead of "$x \leftarrow U(S)$." If inp is a string, then "out $\leftarrow \mathsf{A}(\mathsf{inp})$" denotes the output of algorithm A when run on input inp. If A is deterministic, then out is a fixed value and we write "out $:= \mathsf{A}(\mathsf{inp})$." We also use the notation "out $:= \mathsf{A}(\mathsf{inp}; r)$" to make the randomness r explicit.

For an odd positive integer q, we define $r' := r \bmod^{\pm} q$ to be the unique element $r' \in [-(q-1)/2, (q-1)/2]$ with $r' \equiv r \pmod{q}$.

2.2 Public-Key Encryption (PKE)

The model for PKE schemes is summarized as follows:

Definition 2.1. *A PKE scheme* PKE *consists of the following triple of polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$*:*

- $\mathsf{Gen}(1^\lambda; r_g) \rightarrow (pk, sk)$*: a key-generation algorithm that takes as input* 1^λ*, where* λ *is the security parameter, and randomness* $r_g \in \mathcal{R}_{\mathsf{Gen}}$ *and outputs a pair of keys* (pk, sk)*.* pk *and* sk *are called the encryption key and decryption key, respectively.*
- $\mathsf{Enc}(pk, m; r_e) \rightarrow ct$*: an encryption algorithm that takes as input encryption key* pk*, message* $m \in \mathcal{M}$*, and randomness* $r_e \in \mathcal{R}_{\mathsf{Enc}}$ *and outputs ciphertext* $ct \in \mathcal{C}$*.*
- $\mathsf{Dec}(sk, ct) \rightarrow m/\bot$*: a decryption algorithm that takes as input decryption key* sk *and ciphertext* ct *and outputs message* $m \in \mathcal{M}$ *or a rejection symbol* $\bot \notin \mathcal{M}$*.*

Definition 2.2. *We say a PKE scheme* PKE *is deterministic if* Enc *is deterministic. DPKE stands for deterministic public-key encryption.*

We omit the definition of the correctness.

Plaintext-checking oracle: Since we review and propose key-recovery attacks using plaintext-checking oracle (PCO), we formally define the plaintext-checking oracle [OP01, ABP15].

Definition 2.3 (Plaintext-Checking Oracle). *A plaintext-checking oracle takes as input a plaintext* m *and a ciphertext* ct *and outputs* $\mathsf{boole}(m = \mathsf{Dec}(sk, ct))$*.*

2.3 Key Encapsulation Mechanism (KEM)

The model for KEM schemes is summarized as follows:

Definition 2.4. *A KEM scheme* KEM *consists of the following triple of polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$*:*

- $\mathsf{Gen}(1^\lambda; r_g) \rightarrow (pk, sk)$*: a key-generation algorithm that takes as input* 1^λ*, where* λ *is the security parameter, and randomness* $r_g \in \mathcal{R}_{\mathsf{Gen}}$ *and outputs a pair of keys* (pk, sk)*.* pk *and* sk *are called the encapsulation key and decapsulation key, respectively.*
- $\mathsf{Encaps}(pk; r_e) \rightarrow (ct, K)$*: an encapsulation algorithm that takes as input encapsulation key* pk *and randomness* $r_e \in \mathcal{R}_{\mathsf{Encaps}}$ *and outputs ciphertext* $ct \in \mathcal{C}$ *and key* $K \in \mathcal{K}$*.*
- $\mathsf{Decaps}(sk, ct) \rightarrow K/\bot$*: a decapsulation algorithm that takes as input decapsulation key* sk *and ciphertext* ct *and outputs key* K *or a rejection symbol* $\bot \notin \mathcal{K}$*.*

Key-mismatch oracle: We review the key-mismatch oracle, which is an analogue of the plaintext-checking oracle for PKE.

Definition 2.5 (Key-Mismatch Oracle). *A key-mismatch oracle takes as input a key K and a ciphertext ct and outputs* $\mathrm{boole}(K = \mathsf{Decaps}(sk, ct))$.

3 Variants of the Fujisaki-Okamoto Transformation

We review the variants of the FO transformation that are used by NIST PQC Round 3 candidate KEMs: $\mathsf{FO}^{\not\perp}$ in this section and $\mathsf{FO}^{\not\perp\prime}$, , $\mathsf{HFO}^{\perp}$, $\mathsf{HFO}^{\not\perp}$, SXY, and SXY-KC in subsection B.3.

Let $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a PKE, whose ciphertext space is $\mathcal{C}_{\mathsf{PKE}}$. If PKE is probabilistic, then $\mathcal{R}_{\mathsf{Enc}}$ denotes the randomness space of Enc. Let $\{0,1\}^{k(\lambda)}$ be the key space of KEM.

3.1 FO with implicit rejection, $\mathsf{FO}^{\not\perp}$

$\mathsf{FO}^{\not\perp}$ transforms a weakly-secure probabilistic PKE into IND-CCA-secure KEM, where the identifier $^{\not\perp}$ implies implicit rejection [HHK17]. This variant is used by BIKE and SIKE.

Let $\{0,1\}^{\ell(\lambda)}$ be the plaintext space of PKE. Let $\mathsf{G}\colon \{0,1\}^* \to \mathcal{R}_{\mathsf{Enc}}$ and $\mathsf{KDF}\colon \{0,1\}^{\ell(\lambda)} \times \mathcal{C}_{\mathsf{PKE}} \to \{0,1\}^{k(\lambda)}$ be hash functions modeled by the random oracles. The $\mathsf{FO}^{\not\perp}$ is summarized as Figure 1. Assuming the IND-CPA security of PKE, the obtained KEM scheme is IND-CCA-secure in the QROM (see e.g., [KSS+20]).

$\mathsf{Gen}(1^\lambda)$	$\mathsf{Encaps}(pk)$	$\mathsf{Decaps}(\overline{sk}, ct)$, where $\overline{sk} = (sk, pk, z)$
$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$	$m \leftarrow \{0,1\}^{\ell(\lambda)}$	$m' \leftarrow \mathsf{Dec}(sk, ct)$
$s \leftarrow \{0,1\}^{\ell(\lambda)}$	$r \leftarrow \mathsf{G}(m)$ // for BIKE	$r' \leftarrow \mathsf{G}(m')$ // for BIKE
$\overline{sk} \leftarrow (sk, pk, s)$	$r \leftarrow \mathsf{G}(m, pk)$ // for SIKE	$r' \leftarrow \mathsf{G}(m', pk)$ // for SIKE
return $(pk, \overline{sk})$	$ct \leftarrow \mathsf{Enc}(pk, m; r)$	$ct' \leftarrow \mathsf{Enc}(pk, m'; r')$
	$K \leftarrow \mathsf{KDF}(m, ct)$	**if** $ct = ct'$, **then return** $K \leftarrow \mathsf{KDF}(m', ct)$
	return (K, ct)	**else return** $K \leftarrow \mathsf{KDF}(s, ct)$

Fig. 1: $\mathsf{KEM} := \mathsf{FO}^{\not\perp}[\mathsf{PKE}, \mathsf{G}, \mathsf{KDF}]$ for BIKE and SIKE.

Remark 3.1. BIKE and SIKE do *not* test whole re-encryption check. Roughly speaking, their encryption algorithm Enc is separable into two algorithms Enc_1 and Enc_2. Enc_1 takes pk and randomness r and outputs c_1 and $k \in \{0,1\}^{\ell(\lambda)}$. Enc_2 takes m and k and outputs $c_2 = k \oplus m$.

Using this property, BIKE omits the re-encryption check. Concretely speaking, k in BIKE's Enc_1 is computed as $k \leftarrow \mathsf{H}(r)$, where H is a hash function modeled by the random oracle. BIKE's Dec internally obtains r' and checks the validity of c_1. It then retrieves $m' = c_2 \oplus \mathsf{H}(r')$ and checks the validity of the ciphertext by checking $r' = \mathsf{G}(m')$ or not.

SIKE's Decaps performs the test $c_1' = c_1$ but omits the test $c_2' = c_2$. Since Dec retrieves $m' = c_2 \oplus k$ *deterministically*, we do not need to check the equality of c_2 and c_2'.

4 Key-Recovery Attack against ntrulpr of NTRU Prime

We propose a new key-recovery attack using plaintext-checking oracle against ntrulpr of NTRU Prime [BBC+20]. NTRU LPRime (ntrulpr) is a variant of the LPR PKE [LPR10], which is also based on the Lindner–Peikert PKE [LP11], and has a similar structure to Kyber and Saber. We mimic the KR-PCA against Kyber and Saber proposed by Băetu et al. [BDH+19] and Huguenin-Dumittan and Vaudenay [HV20].

Table 3: Parameter sets of ntrulpr of NTRU Prime

parameter sets	p	q	w	δ	τ_0	τ_1	τ_2	τ_3
`ntrulpr653`	653	4621	252	289	2175	113	2031	290
`ntrulpr761`	761	4591	250	292	2156	114	2007	287
`ntrulpr857`	857	5167	281	329	2433	101	2265	324
`ntrulpr953`	953	6343	345	404	2997	82	2798	400
`ntrulpr1013`	1013	7177	392	450	3367	73	3143	449
`ntrulpr1277`	1277	7879	429	502	3724	66	3469	496

Table 4: The PCO's behaviors

sk_i	$\mathsf{PCO}(\vec{1}_{256}, ct_0)$	$\mathsf{PCO}(\vec{1}_{256}, ct_1)$
1	1	1
0	1	0
-1	0	0

ntrulpr of NTRU Prime: NTRU LPRime has parameter sets $p, q, w, \delta, \tau_0, \tau_1, \tau_2$, and τ_3. We note that $q = 6q' + 1$ for some q' and $q \geq 16w + 2\delta + 3$. For concrete values, see Table 3.

Let $\mathcal{R} := \mathbb{Z}[x]/(x^p - x - 1)$ and $\mathcal{R}_q := (\mathbb{Z}/q)[x]/(x^p - x - 1)$. Let $\mathcal{S} := \{a = \sum_{i=0}^{p-1} a_i x^i \in \mathcal{R} \mid a_i \in \{-1, 0, +1\}, \mathsf{HW}(a) = w\}$, a set of "short" polynomials.

For $a \in [-(q-1)/2, (q-1)/2]$, define $\mathsf{Round}(a) = 3 \cdot \lceil a/3 \rfloor$.[10] For a polynomial $A = \sum_i a_i x^i \in \mathcal{R}_q$, we define $\mathsf{trunc}(A, l) = (a_0, \ldots, a_{l-1}) \in \mathbb{Z}_q^l$. For $C \in [0, q)$, define $\mathsf{Top}(C) = \lfloor (\tau_1(C + \tau_0) + 2^{14})/2^{15} \rfloor$. For $T \in [0, 16)$, define $\mathsf{Right}(T) = \tau_3 T - \tau_2 \in \mathbb{Z}_q$. For $a \in \mathbb{Z}$, define $\mathsf{Sign}(a) = 1$ if $a < 0$, 0 otherwise.

The underlying CPA-secure PKE scheme[11] works as follows:

- $\mathsf{Gen}(pp)$: Generate $A \leftarrow \mathcal{R}_q$ and $sk \leftarrow \mathcal{S}$. Output $pk = (A, B = \mathsf{Round}(A \cdot sk))$.
- $\mathsf{Enc}(pk, \mu \in \{0,1\}^{256})$: Choose $t \leftarrow \mathcal{S}$ and output

$$(U, V) \leftarrow \big(\mathsf{Round}(t \cdot A), \mathsf{Top}(\mathsf{trunc}(t \cdot B, 256) + \mu(q-1)/2)\big).$$

- $\mathsf{Dec}(sk, (U, V))$: Compute $r = \mathsf{Right}(V) - \mathsf{trunc}(sk \cdot U, 256) + (4w + 1) \cdot \vec{1}_{256} \in \mathbb{Z}^{256}$ and outputs $m = \mathsf{Sign}(r \bmod^{\pm} q)$.

4.1 Key-Recovery Attack

We mainly follow the KR-PCAs against Kyber and Saber in Baetu et al. [BDH+19] and Huguenin-Dumittan and Vaudenay [HV20], but we need some tweaks. Roughly speaking, to determine the i-th coefficient of sk, their attack queries $(a, b \cdot x^i)$ with constant a and b and a candidate plaintext, because in the case of Kyber and Saber, the dimension of V is the same as that of the base ring. However, ntrulpr truncates tB to reduce redundancy, so we need to modify the query ciphertext. Note that we can *shift* the effect of sk_i into *constant coefficient* by multiplying x^{p-i}. That is, for $i = 1, \ldots, p-1$, we have

$$\begin{aligned} x^{p-i} \cdot a \\ &= a_i + (a_i + a_{i+1})x + (a_{i+1} + a_{i+2})x^2 + \cdots + (a_{p-2} + a_{p-1})x^{p-i-1} + (a_{p-1} + a_0)x^{p-i} \\ &\quad + a_1 x^{p-i+1} + a_2 x^{p-i+2} + \cdots + a_{i-1} x^{p-1}. \end{aligned}$$

Using this relation, we show the following two lemmas:

Lemma 4.1 (For general $i \in [1, p)$). *Let $c = \tau_2 - (4w + 1)$, $b = \lfloor (c-1)/6 \rfloor \cdot 3$ and $t_\beta = \lfloor (\beta b + c - 1)/\tau_3 \rfloor$ for $\beta \in \{0, 1\}$. Let us consider our test ciphertext $ct_j = (b \cdot x^{p-i}, (t_\beta, 0, \ldots, 0))$ for $\beta \in \{0, 1\}$ and candidate plaintext $\vec{1}_{256}$. Then, we have the relations between the i-th coordinate of decryption key and the behavior of PCO as in Table 4.*

[10] When $q = 6q' + 1$, $\mathsf{Round}([-(q-1)/2, (q-1)/2]) \in [-(q-1)/2, (q-1)/2]$.

[11] 'NTRU LPRime Core' in the specification.

Proof. The decryption algorithm computes $r = \mathsf{Right}\big((t_\beta, 0, \ldots, 0)\big) - \mathsf{trunc}\big(sk \cdot b \cdot x^{p-i}, 256\big) + (4w+1) \cdot \vec{1}_{256}$. Expanding this, we have

$$\begin{cases} r_0 = \tau_3 t_\beta - b \cdot sk_i - c, & \\ r_j = -b \cdot (sk_{i+j-1 \bmod p} + sk_{i+j \bmod p}) - c & (j = 1, 2, \ldots, \min\{256, p-i\}) \\ r_j = -b \cdot sk_{j-(p-i) \bmod p} - c & (j = p-i+1, \ldots, \min\{256, p-1\}). \end{cases}$$

Recall that $sk_i \in \{-1, 0, +1\}$ for all i since sk is in $\mathcal{S}$. Thus, we have $r_j \in \{-2b-c, -b-c, -c, b-c, 2b-c\}$ for $j = 1, \ldots, 256$. Since we set $b = \lfloor (c-1)/6 \rfloor \cdot 3 \le (c-1)/2$, we have $-2b-c > -2c$ and $2b-c < 0$. Fortunately, we have $-2c = -2\tau_2 - 8w - 2 \ge -(q-1)/2$ for all parameter sets. Thus, r_j's are decoded into 1 for $j = 1, \ldots, 256$.

Let us consider r_0. We have

$$r_0 = \tau_3 t_\beta - b \cdot sk_i - c > 0 \iff (\tau_3 t_\beta - c)/b > sk_i$$

By our setting, if $t_\beta = t_0$ (and t_1), then $(\tau_3 t_\beta - c)/b$ is slightly smaller than 0 (and 1) for all parameter sets, respectively. In addition, we have $\tau_3 t_1 + b - c \le (q-1)/2$ for all parameter sets. Therefore, r_0 for t_0 is decoded into 0 if and only if $sk_i < 0$ and r_0 for t_1 is decoded into 0 if and only if $sk_i < 1$. This completes the proof. □

By a similar argument, we have the following lemma on sk_0.

Lemma 4.2 **($i = 0$).** *Let $c = \tau_2 - (4w+1)$, $b = \lceil (c-1)/6 \rceil \cdot 3$ and $t_\beta = \lfloor (\beta b + c - 1)/\tau_3 \rfloor$ for $\beta \in \{0, 1\}$. Let us consider our test ciphertext $ct_\beta = (b, (t_\beta, 0, \ldots, 0))$ and candidate plaintext $\vec{1}_{256}$. Then, we have the relations between the constant term of decryption key and the behavior of PCO as in Table 4.*

Using the above lemmas, we can determine sk_i for $i = 0, \ldots, p-1$ by testing $2p$ queries with the PCO.

4.2 Trade-Off

We observe that an adversary can obtain $K' = \mathsf{KDF}(m', ct)$ by skipping the equality test instead of the equality of K' and K_{guess} or the equality of m' and m_{guess}. Therefore, the adversary can check if $m' = m_{\text{guess}}$ or not by computing $K_{\text{guess}} = \mathsf{KDF}(m_{\text{guess}}, ct)$ by itself. This enables the adversary to determine ℓ coefficients of the secret key at once by sacrificing the computational efficiency.

For simplicity, we let $\ell = 2^k < 256$.

Determine $sk_{y\ell}, \ldots, sk_{y\ell+\ell-1}$ for $y = 0, \ldots, 256/\ell - 1$: Let us determine ℓ coefficients $sk_{y\ell}, \ldots, sk_{y\ell+\ell-1}$ of sk at once, where $y = 0, \ldots, 256/\ell - 1$. Suppose that we query two ciphertexts

$$ct_\beta = (U, V_\beta) = \big(b, (\overbrace{0, \ldots, 0}^{y\ell}, \overbrace{t_\beta, \ldots, t_\beta}^{\ell}, \overbrace{0, \ldots, 0}^{256-(y+1)\ell})\big)$$

for $\beta \in \{0, 1\}$. The decryption algorithm computes $r = \mathsf{Right}(V_\beta) - \mathsf{trunc}(sk \cdot b, 256) + (4w+1) \cdot \vec{1}_{256}$. Expanding this, we have

$$r_j = \begin{cases} \tau_3 t_\beta - b \cdot sk_j - c & (j = y\ell, \ldots, y\ell + \ell - 1) \\ -b \cdot sk_j - c & (j = 0, \ldots, y\ell - 1, (y+1)\ell, \ldots, 256). \end{cases}$$

By using the argument in the proof of Lemma 4.1, r_j's are decoded into 1 for $j = 0, \ldots, y\ell - 1, (y+1)\ell, \ldots, 256$. We also have, for $j = y\ell, \ldots, y\ell + \ell - 1$, r_j for t_1 is decoded into 0 if and only if $sk_i < 0$ and r_j for t_2 is decoded into 0 if and only if $sk_i < 1$.

Seeing $K = \mathsf{KDF}(m', ct_\beta)$ where $m' = \mathsf{Dec}(sk, ct_\beta)$, we compute $K_{\text{guess}} = \mathsf{KDF}(m_{\text{guess}}, ct_\beta)$ for $m_{\text{guess}} = \vec{1}_{y\ell} \| m'' \| \vec{1}_{256-(y+1)\ell}$ for all $m'' \in \{0, 1\}^\ell$ and determine sk_j for $j = y\ell, \ldots, y\ell + \ell - 1$.

Determine $sk_{y\ell}, \ldots, sk_{y\ell+\ell-1}$ for $y = 256/\ell, \ldots, \lfloor p/\ell \rfloor$: Suppose that we have determined $y\ell$ coefficients $sk_0, \ldots, sk_{y\ell-1}$ for some $y \in \{256/\ell, \ldots, \lfloor p/\ell \rfloor\}$. Let us determine ℓ coefficients $sk_{y\ell}, \ldots, sk_{y\ell+\ell-1}$ at once: Let $t_\beta = \lfloor (\beta b + c - 1)/\tau_3 \rfloor$ for $\beta \in \{-1, 0, 1, 2\}$. Suppose that we query four ciphertexts

$$ct_\beta = (U, V_\beta) = \big(b \cdot x^{p-y\ell-1}, (0, \overbrace{t_\beta, \ldots, t_\beta}^{\ell}, \overbrace{0, \ldots, 0}^{256-\ell-1})\big)$$

for $\beta \in \{-1,0,1,2\}$. The decryption algorithm computes $r = \mathrm{Right}(V_\beta) - \mathrm{trunc}(sk \cdot bx^{p-y\ell-1}, 256) + (4w+1) \cdot \vec{1}_{256}$. Expanding this, we have

$$r_j = \begin{cases} -b \cdot sk_{y\ell-1} - c & (j=0) \\ \tau_3 t_\beta - b \cdot (sk_{y\ell+j-2 \bmod p} + sk_{y\ell+j-1 \bmod p}) - c & (j = 1,2,\dots,\ell) \\ -b \cdot (sk_{y\ell+j-2 \bmod p} + sk_{y\ell+j-1 \bmod p}) - c & (j = \ell+1,\dots,\min\{256, p-y\ell\}) \\ -b \cdot sk_{j-(p-i) \bmod p} - c & (j = \min\{256, p-y\ell+1\},\dots,256). \end{cases}$$

By using the argument in the proof of Lemma 4.1, r_j's are decoded into 1 for $j = 0$ and $j = \ell+1,\dots,256$.

Let us consider r_j for $j = 1,\dots,\ell$. We have

$$r_j = \tau_3 t_\beta - b \cdot (sk_j + sk_{j+1}) - c > 0 \iff (\tau_3 t_\beta - c)/b > sk_j + sk_{j+1}.$$

By our setting, $(\tau_3 t_\beta - c)/b$ is slightly smaller than β for all parameter sets, respectively. In addition, we have $-(q-1)/2 \le \tau_3 t_\beta - 2b - c$ and $\tau_3 t_\beta + 2b - c \le (q-1)/2$ for all parameter sets. Therefore, r_j for t_β is decoded into 0 if and only if $sk_i < \beta$.

Seeing $K' = \mathsf{KDF}(m', ct_\beta)$ where $m' = \mathsf{Dec}(sk, ct_\beta)$, we compute $K_{\mathrm{guess}} = \mathsf{KDF}(m_{\mathrm{guess}}, ct_\beta)$ for $m_{\mathrm{guess}} = 1\|m''\|\vec{1}_{256-\ell-1}$ for all $m'' \in \{0,1\}^\ell$ and determine $sk_{y\ell+j-2} + sk_{y\ell+j-1} \in \{-2,-1,0,1,2\}$ for $j = 1,\dots,\ell$. Since we know $sk_{y\ell-1}$, we can determine $sk_{y\ell},\dots,sk_{y\ell+\ell-1}$ sequentially.

5 Skipping the Equality Test

In this section, we describe the fault-injection attack on the equality checking of each KEM implementation. First, we examine the implementation of pqm4 [KRSS] for each scheme and discuss the possibility of skipping the equivalence test. To identify the instructions to be skipped, we cross-compiled the C code in pqm4 with GCC 8.3.1 running on Debian bullseye. The compilation options were basically according to pqm4, with "-O3" as an optimization option.

We do not mention the attacks on Classic McEliece and HQC in this section because pqm4 does not include their ARM Cortex M4-optimized code. Hereafter, we describe the possibility of skip attacks on NTRU Prime, FrodoKEM, Kyber, Saber, NTRU, BIKE, and SIKE.

If the reader is unfamiliar to Arm Coretex M4, please see the manual [12].

5.1 NTRU Prime – CCA Bug

The functions in the C code related to the FO-like transformation are `crypto_kem_dec`, `Decap`, and `Ciphertexts_diff_mask`.[13] Figure 2 shows the source code of NTRU Prime's comparison in pqm4. Note that we omit the `crypto_kem_dec` function as it just calls `Decap`.

Let us consider how `Ciphertexts_diff_mask` computes the return value. It initializes the `uint16` variable `differentbits` as `0`. After some computations, it outputs `((-1)-((differentbits-1)>>31))` in line 17. The value is initialized as `0` and *unchanged* before the return value is computed; these computations only involve `differentbits2`. Thus, we eventually obtain `0` as the result of `(-1)-((0-1)>>31)` and `ciphertexts_diff_mask` always outputs `0`.

`Decap` first decrypts $r = \mathrm{Dec}(sk, c)$ and re-encrypts it into `r_enc` and `cnew`. In line 15, `mask` is always `0`, since `Ciphertexts_diff_mask` always returns `0` as we explained. Thus, `r_enc`, which is the result of faulty decryption, is unchanged, and `Decap` always sets `k` as the result of `H(1,r_enc,c)`. This means that there is no re-encryption check and the implementation opens the attack surface of chosen-ciphertext attacks.

5.2 FrodoKEM – Timing Attack

The decapsulation of FrodoKEM is performed in the `crypto_kem_dec` function.[14] Figure 3 shows the source code of the equality test in the function. From the source code, this function uses the `memcmp` function to compare the ciphertext and the re-encryption result. This indicates that the current implementation is still vulnerable to the timing attack by Guo et al. [GJN20].

[12] https://developer.arm.com/documentation/100166/0001. See https://developer.arm.com/documentation/100166/0001/Programmers-Model/Instruction-set-summary/Table-of-processor-instructions?lang=en for instruction set.

[13] The source code of these functions is https://github.com/mupq/pqm4/blob/master/crypto_kem/sntrup761/m4f/kem.c.

[14] https://github.com/mupq/pqm4/blob/master/crypto_kem/frodokem640shake/m4/kem.c

```
static int Ciphertexts_diff_mask(const unsigned char *c,
                                 const unsigned char *c2)
{
  uint16 differentbits = 0;
  int len = Ciphertexts_bytes+Confirm_bytes;

  int *cc = (int *)(void *)c;
  int *cc2 = (int *)(void *)c2;
  int differentbits2 = 0;
  for (len-=4 ;len>=0; len-=4) {
    differentbits2 = __USADA8((*cc++),(*cc2++),differentbits2);
  }
  c = (unsigned char *)(void *) cc;
  c2 = (unsigned char *)(void *) cc2;
  for (len &= 3; len > 0; len--)
    differentbits2 =__USADA8((*c++),(*c2++),differentbits2);
  return ((-1)-((differentbits-1)>>31));
}
```

```
static void Decap(unsigned char *k,const unsigned char *c,
                                 const unsigned char *sk)
{
  const unsigned char *pk = sk + SecretKeys_bytes;
  const unsigned char *rho = pk + PublicKeys_bytes;
  const unsigned char *cache = rho + Inputs_bytes;
  Inputs r;
  unsigned char r_enc[Inputs_bytes];
  unsigned char cnew[Ciphertexts_bytes+Confirm_bytes];
  int mask;
  int i;

  ZDecrypt(r,c,sk);
  Hide(cnew,r_enc,r,pk,cache);
  mask = Ciphertexts_diff_mask(c,cnew);
  for (i = 0;i < Inputs_bytes;++i) r_enc[i] ^= mask&(r_enc[i]^rho[i]);
  HashSession(k,1+mask,r_enc,c);
}
```

Fig. 2: NTRU Prime's comparison in pqm4.

```
    // Is (Bp == BBp & C == CC) = true
    if (memcmp(Bp, BBp, 2 * PARAMS_N * PARAMS_NBAR) == 0 &&
          memcmp(C, CC, 2 * PARAMS_NBAR * PARAMS_NBAR) == 0) {
        // Load k' to do ss = F(ct || k')
        memcpy(Fin_k, kprime, CRYPTO_BYTES);
    } else {
        // Load s to do ss = F(ct || s)
        memcpy(Fin_k, sk_s, CRYPTO_BYTES);
    }
    shake(ss, CRYPTO_BYTES, Fin, CRYPTO_CIPHERTEXTBYTES + CRYPTO_BYTES);
```

Fig. 3: Frodo's comparison in pqm4.

```
1      int crypto_kem_dec(uint8_t *k, const uint8_t *c, const uint8_t *sk)
2  {
3      uint8_t fail;
4      uint8_t buf[64];
5      uint8_t kr[64]; // Will contain key, coins
6      const uint8_t *pk = sk + SABER_INDCPA_SECRETKEYBYTES;
7      const uint8_t *hpk = sk + SABER_SECRETKEYBYTES - 64;
8                                  // Save hash by storing h(pk) in sk
9
10     indcpa_kem_dec(sk, c, buf);
11     memcpy(buf + 32, hpk, 32);
12     sha3_512(kr, buf, 64);
13     fail = indcpa_kem_enc_cmp(buf, kr + 32, pk, c);
14     sha3_256(kr + 32, c, SABER_BYTES_CCA_DEC);
15     cmov(kr, sk + SABER_SECRETKEYBYTES - SABER_KEYBYTES,
16                                               SABER_KEYBYTES, fail);
17     sha3_256(k, kr, 64);
18     return (0);
19 }
20
```

Fig. 4: Saber's comparison in pqm4

5.3 Kyber, Saber, and NTRU – cmov

In this subsection, we describe the skip attacks on Kyber, Saber, and NTRU among the finalists. The basic idea of the skip attacks on these implementations is identical, and thus we describe the case of Saber as an example to explain the skip attack procedure. Figure 4 shows the `crypto_kem_dec` function that performs the decapsulation of FO transformation[15].

The `crypto_kem_dec` function performs re-encryption using the `indcpa_kem_enc_cmp` function at line 13 and stores the comparison results of the ciphertext and the re-encryption result into a variable `fail`. If these ciphertexts are not the same, `fail` becomes 1 and, if they are the same, `fail` becomes `0`. At line 15, the `cmov` substitutes a random value for `kr` when `fail` is 1. Note here that the hash value calculated from the decrypted result is stored in the variable `kr` before `cmov` is called, and this means that we can perform a key-recovery attack by skipping the call of `cmov` even when `fail` is 1.

[15] https://github.com/mupq/pqm4/blob/master/crypto_kem/saber/m4f/kem.c

```
    bl   sha3_256
.LVL26:
.loc 1 79 3 is_stmt 1 view .LVU62
    uxtb    r3, r7
    add r1, r4, #1536
    add r0, sp, #64
    movs    r2, #32
    bl   cmov
.LVL27:
    .loc 1 82 3 view .LVU63
    movs    r2, #64
    mov r0, r6
    add r1, sp, r2
    bl   sha3_256
```

Fig. 5: Assembly code of Saber's comparison in pqm4

Figure 5 shows the assembly code corresponding to the call of cmov. This program first calls the sha3_256 function at line 1, prepares the arguments of cmov at line 4–7, calls cmov at line 8, and finally prepares the arguments and call the sha3_256 function at line 10–14. From this code, we notice that Saber can be attacked by skipping `bl cmov` at line 8 using fault injection. In addition to Saber, NTRU and Kyber also use cmov in the same manner, and therefore, this attack can be applied to all of them.

5.4 BIKE – For loop

We describe the skip attack on BIKE in this subsection. Figure 6 shows the C code of BIKE's comparison in the decapsulation[16]. We also show secure_cmp function and secure_l32_mask function in Figure 7. Line 4–5 in Figure 6 compares the hash value of the original message and that of the decrypted message from the ciphertext. Then, if they are equivalent, the for block at line 9–12 stores the decrypted message into m_prime.raw[i]. Therefore, the goal of the fault-injection attack is to store the decrypted message even when these hash values are not equivalent. For this purpose, we need to force the variable mask to be 0.

Figure 8 shows the assembly code corresponding to the line 5–7 in the C code to explain the position of a fault injection. Line 1–30 and line 31–44 in the assembly code correspond to line 5 and line 7 in the C code, respectively. The operation we need to skip for a key-recovery attack is `ldr r2, [sp, #20]` at line 33 in this assembly code for the following reason.

Before line 33 in the assembly code, the r2 register is used in `cmp r2, #0` at line 26. This corresponds to `return (0 == res);` at line 11 in secure_cmp function (Figure 7). Therefore, at this time, the r2 register contains the value of the res variable. The value of the r2 register does not become 0 from the attack assumption because the value of the res variable is not 0 when the two arguments of secure_cmp are not equivalent. Thus, the r2 register must be a non-zero value if line 33 in the assembly code is skipped. After line 33, the value of the r2 register is used at line 41. This line corresponds to line 8 in the secure_l32_mask function (Figure 7). The secure_l32_mask function compares the two arguments v1 and v2 and returns 0 when `v1 < v2` holds. mask becomes 0 when v2 is not 0 because v1 is 0 as shown in Figure 6. Meanwhile, we note that the variable v2 does not become 0 when we skip line 33 in the assembly code because the r2 register corresponds to the v2 variable. From the above, we can fix mask to 0 by the fault injection, and thus the key-recovery attack is possible.

```
1  // Check if H(m') is equal to (e0', e1')
2  // (in constant-time)
3  GUARD(function_h(&e_tmp, &m_prime));
4  success_cond = secure_cmp(PE0_RAW(&e_prime), PE0_RAW(&e_tmp), R_BYTES);
5  success_cond &= secure_cmp(PE1_RAW(&e_prime), PE1_RAW(&e_tmp), R_BYTES);
6
7  // Compute either K(m', C) or K(sigma, C) based on the success condition
8  mask = secure_l32_mask(0, success_cond);
9  for(size_t i = 0; i < M_BYTES; i++) {
10   m_prime.raw[i] &= u8_barrier(~mask);
11   m_prime.raw[i] |= (u8_barrier(mask) & l_sk.sigma.raw[i]);
12 }
```

Fig. 6: BIKE's comparison in pqm4.

5.5 SIKE – Simple If

This subsection describe the skip attack on SIKE. Figure 9 shows the C code and its assembly of the comparison process in the FO transformation[17].

The target of fault injection in C code is the if statement at line 3–4. The assembly code in Figure 9 corresponds to the if statement. The process of condition in the if statement at line 3 in the C code corresponds

[16] https://github.com/mupq/pqm4/blob/master/crypto_kem/bikel1/m4f/kem.c

[17] https://github.com/mupq/pqm4/blob/master/crypto_kem/sikep434/m4/sike.inc

```c
_INLINE_ uint32_t secure_cmp(IN const uint8_t *a,
                             IN const uint8_t *b,
                             IN const uint32_t size)
{
  volatile uint8_t res = 0;

  for(uint32_t i = 0; i < size; ++i) {
    res |= (a[i] ^ b[i]);
  }

  return (0 == res);
}
```

```c
// Return 0 if v1 < v2, (-1) otherwise
_INLINE_ uint32_t secure_l32_mask(IN const uint32_t v1,
                                  IN const uint32_t v2)
{
  // If v1 >= v2 then the subtraction result is 0^32||(v1-v2).
  // else it is 1^32||(v2-v1+1). Subsequently, negating the upper
  // 32 bits gives 0 if v1 < v2 and otherwise (-1).
  return ~((uint32_t)(((uint64_t)v1 - (uint64_t)v2) >> 32));
}
```

Fig. 7: `secure_cmp` and `secure_l32_mask` function of BIKE in pqm4.

to line 1-4 in the assembly code. In the assembly code, “`bl memcmp`” compares the variables `c0_` and `ct`. If they are not equivalent, “`cbnz r0, .L500`” performs a jump to line 28. Note that, even if we jump to line 28, the procedure comes back to line 5 because of “`b .L495`” at line 40. In other words, line 28–40 in the assembly code correspond to the process in the `if` block at line 4 in the C code. Thus, we can perform the skip attack on SIKE by injecting a fault into `cbnz r0, .L500` at line 3.

```
.L26:
    .loc 1 69 5 is_stmt 1 view .LVU627
    ldrb    r2, [r5, #1]!
.LVL169:
    .loc 1 69 9 view .LVU629
    ldrb    r4, [r1, #1]!
    ldrb    r3, [sp, #18]
    eors    r2, r2, r4
    orrs    r3, r3, r2
    .loc 1 68 3 view .LVU630
    cmp    r0, r5
    .loc 1 69 9 view .LVU631
    strb    r3, [sp, #18]
.LVL170:
    .loc 1 68 3 view .LVU632
    bne    .L26
.LBE629:
    .loc 1 72 3 is_stmt 1 view .LVU633
.LVL171:
    .loc 1 72 13 is_stmt 0 view .LVU634
    ldrb    r2, [sp, #18]
.LBE628:
.LBE627:
    .loc 2 278 16 view .LVU635
    ldr    r3, [sp, #20]
    cmp    r2, #0
    ite    ne
    movne    r3, #0
    andeq    r3, r3, #1
    str    r3, [sp, #20]
    .loc 2 282 3 is_stmt 1 view .LVU636
    .loc 2 282 10 is_stmt 0 view .LVU637
    ldr    r2, [sp, #20]
.LVL172:
.LBB630:
.LBI630:
    .loc 1 113 19 is_stmt 1 view .LVU638
.LBB631:
    .loc 1 140 3 view .LVU639
    .loc 1 140 37 is_stmt 0 view .LVU640
    rsbs    r2, r2, #0
    sbc    r3, r3, r3
    .loc 1 140 10 view .LVU641
    mvns    r5, r3
```

Fig. 8: Assembly code of BIKE’s comparison in pqm4

```
    // Generate shared secret ss <- H(m||ct) or output ss <- H(s||ct)
    EphemeralKeyGeneration_A(ephemeralsk_, c0_);
    if (memcmp(c0_, ct, CRYPTO_PUBLICKEYBYTES) != 0) {
        memcpy(temp, sk, MSG_BYTES);
    }
    memcpy(&temp[MSG_BYTES], ct, CRYPTO_CIPHERTEXTBYTES);
    shake256(ss, CRYPTO_BYTES, temp, CRYPTO_CIPHERTEXTBYTES+MSG_BYTES);
```

```
    bl  memcmp
    .loc 5 88 8 view .LVU4945
    cbnz    r0, .L500
.LVL1930:
.L495:
    .loc 5 91 5 is_stmt 1 view .LVU4946
    mov r1, r4
    add r0, sp, #508
    mov r2, #346
    bl  memcpy
.LVL1931:
    .loc 5 92 5 view .LVU4947
    mov r0, r8
    add r2, sp, #492
    mov r3, #362
    movs    r1, #16
    bl  shake256
.LVL1932:
    .loc 5 94 5 view .LVU4948
    .loc 5 95 1 is_stmt 0 view .LVU4949
    movs    r0, #0
    add sp, sp, #856
.LCFI116:
    .cfi_remember_state
    .cfi_def_cfa_offset 24
    pop {r4, r5, r6, r7, r8, pc}
.LVL1933:
.L500:
.LCFI117:
    .cfi_restore_state
    .loc 5 89 9 is_stmt 1 view .LVU4950
    ldr r0, [r5]
    ldr r1, [r5, #4]
    ldr r2, [r5, #8]
    ldr r3, [r5, #12]
    add r5, sp, #492
.LVL1934:
    .loc 5 89 9 is_stmt 0 view .LVU4951
    stmia   r5!, {r0, r1, r2, r3}
    b   .L495
```

Fig. 9: SIKE's comparison in pqm4

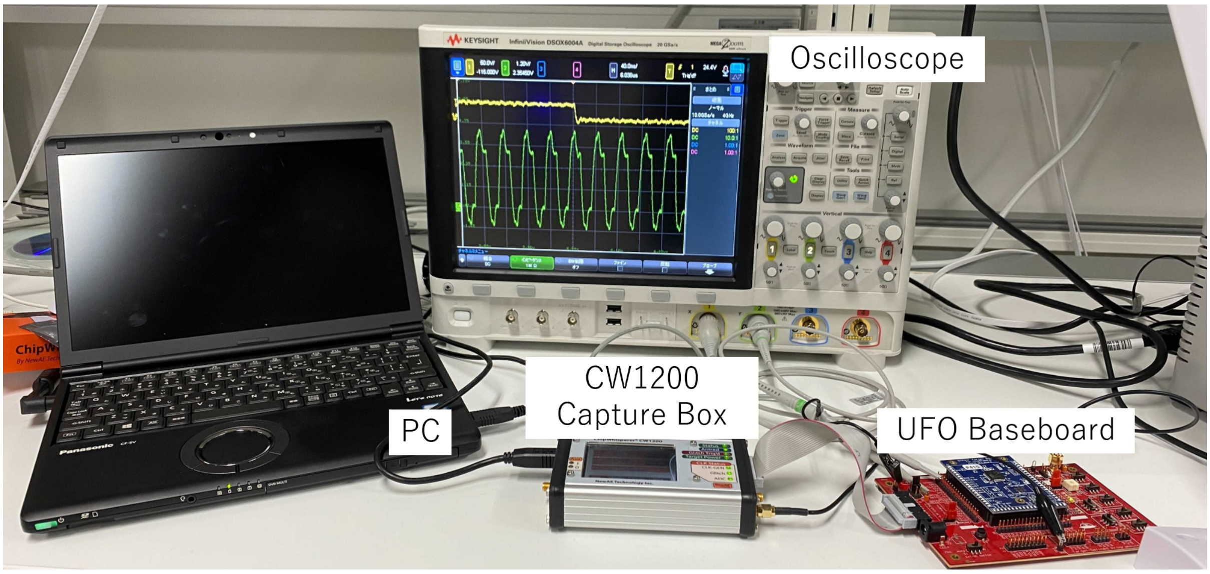

Fig. 10: Experimental setup overview.

Table 5: Numbers of failures and successes when we conducted 100 skip attacks on each scheme

Name	# failures	# Successes	Expected # required queries
Kyber – Kyber512	60	52	5908
NTRU – ntruhps2048509	74	46	2235
Saber – LightSaber	33	33	15515
BIKE – Bikel1	49	34	-
SIKE – sikep434	30	15	1787

6 Experimental Attacks

In this section, we conduct the experimental skip attacks on the pqm4 implementation of the above mentioned KEM schemes. The target schemes in this section are Kyber, NTRU, Saber, BIKE, and SIKE, which were shown to be attackable by a single fault injection in the previous section. In this experiment, we used the parameters of the security level 1 for all schemes.

6.1 Setup

Figure 10 shows the experimental environment. The target chip under attack is an STM32F415 microcontroller with an ARM Cortex M4 core, which is a de-facto standard platform to evaluate software implementation of schemes running in NIST's PQC process. The target device is mounted on a ChipWhisperer cw308 UFO baseboard, which enables us to perform fault-injection attacks using a glitchy clock. The ChipWhisperer cw1200 capture box is used to generate the base clock, and the clock frequency was set to 24 MHz. The glitch parameters for instruction skipping were searched for by sweeping the parameters to find the one that successfully skips the instruction. We use the implementation in pqm4 for each KEM scheme, and "O3" was specified as an optimization option during compilation.

6.2 Results

Table 5 reports the experimental results of the proposed skip attacks. In Table 5, we show the number of times when a fault occurred on the device and the number of successful instruction skips when we performed 100 fault injections for each scheme. Also, the table shows the number of required queries to recover the secret key from

Table 6: Summary of our findings on NIST PQC Round 3 KEM Candidates (finalists and alternatives) and their implementations in pqm4: PCA implies plaintext-checking attack.

Name	Effect of PCA	Attack Surface in pqm4	Effect of FIA in pqm4
Classic McEliece [ABC+20]	Unknown	–	–
Kyber [SAB+20]	Key recovery	Skip	Key recovery
NTRU – ntruhps [CDH+20]	Key recovery	Skip	Key recovery
NTRU – ntruhrss [CDH+20]	Key recovery	Skip	Key recovery
Saber [DKR+20]	Key recovery	Skip	Key recovery
BIKE [ABB+20]	Key leakage (New)	Skip	Key leakage
FrodoKEM [NAB+20]	Key recovery	CCA bug (Timing)	Key recovery
HQC [AAB+20]	Key recovery	–	–
NTRU Prime – sntrupr [BBC+20]	Key recovery	CCA bug	Key recovery
NTRU Prime – ntrulpr [BBC+20]	Key recovery (New)	CCA bug	Key recovery
SIKE [JAC+20]	Key recovery	Skip	Key recovery

each scheme using fault injection.[18][19] These required query numbers are calculated by multiplying the minimum required number of queries for a key-recovery attack and the inverse of the success rate of a skip attack. We only omitted the number of required queries for the case of BIKE in this table because it is difficult to fully recover the secret key. From the table, we confirm that the probability of a successful attack was about 15-50%, and there is a difference in the probability of successful attacks among Saber, Kyber, and NTRU, although the fault-injection capability is almost the same. This would be because of the difference in instructions before and after the call of the `cmov` function that affects the state of pipeline registers in the microcontroller.

In addition, in this experiment, the injected faults did not always cause a single instruction skip as expected and sometimes crashed the device, which led to a non-negligible cost for an oracle access. A similar phenomenon was also observed in [PP21] in fault-injection attacks on lattice KEMs using ChipWhisperer, and more sophisticated equipment for fault injection should achieve higher attack stability.

7 Conclusion

From the viewpoint of fault-injection attacks, we have investigate *all* NIST PQC Round 3 KEM candidates, which use variants of the FO transformation. We survey effective key-recovery attacks if we can skip the equality test.

We found the existing key-recovery attacks against Kyber, NTRU, Saber, FrodoKEM, HQC, and SIKE (Table 2). We have proposed a new key-recovery attack against ntrulpr of NTRU Prime. We also pointed out trade-offs between the number of queries and computational costs when the target is Kyber, Saber, or ntrulpr. We also reported attacks against sntrupr of NTRU Prime and BIKE that lead to leakage of information of secret keys.

The open-source pqm4 library contains Kyber, NTRU, Saber, BIKE, FrodoKEM, NTRU Prime, and SIKE. We show that giving a single instruction-skipping fault in the decapsulation processes leads to skipping the equality test *virtually* for Kyber, NTRU, Saber, BIKE, and SIKE. We also report the implementation of NTRU Prime allows chosen-ciphertext attacks freely and the timing side-channel of FrodoKEM reported in Guo et al. [GJN20] remains.

Finally, we have reported the experimental attacks against Kyber, NTRU, Saber, BIKE, and SIKE on pqm4.

References

AAA+20. Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status report on the second round of the NIST post-quantum cryptography standardization process, July 2020. https://csrc.nist.gov/publications/detail/nistir/8309/final. 2

[18] In practice, we may need more queries than the values shown in the table, because the value of the secret key may occasionally carry an error due to an inserted fault. For simplicity, we ignore such situations here.

[19] On Saber and Kyber, we have trade-offs between the number of expected queries and efficiency. In this table, we use $\ell = 1$.

AAB+20. Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and Jurjen Bos. HQC. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. 3, 4, 17, 33

ABB+20. Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, Valentin Vasseur, and Santosh Ghosh. BIKE. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. 3, 4, 17, 31

ABC+20. Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. 3, 4, 17, 25, 26

ABGV08. Ali C. Atıcı, Lejla Batina, Benedikt Gierlichs, and Ingrid Verbauwhede. Power analysis on NTRU implementations for RFIDs: First results. The 4th Workshop on RFID Security – RFIDSec 2008, July 9th–11th, Budapest, 2008. https://www.esat.kuleuven.be/cosic/publications/article-1134.pdf. 29

ABM+03. Adrian Antipa, Daniel R. L. Brown, Alfred Menezes, René Struik, and Scott A. Vanstone. Validation of elliptic curve public keys. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 211–223. Springer, Heidelberg, January 2003. 2

ABP15. Michel Abdalla, Fabrice Benhamouda, and David Pointcheval. Public-key encryption indistinguishable under plaintext-checkable attacks. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 332–352. Springer, Heidelberg, March / April 2015. 5

AD97. Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *29th ACM STOC*, pages 284–293. ACM Press, May 1997. 2

AM09. Divesh Aggarwal and Ueli Maurer. Breaking RSA generically is equivalent to factoring. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 36–53. Springer, Heidelberg, April 2009. 2

AR21. Amund Askeland and Sondre Rønjom. A side-channel assisted attack on NTRU. Cryptology ePrint Archive, Report 2021/790, 2021. https://eprint.iacr.org/2021/790. 29

BBC+20. Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, Chitchanok Chuengsatiansup, Tanja Lange, Adrian Marotzke, Bo-Yuan Peng, Nicola Tuveri, Christine van Vredendaal, and Bo-Yin Yang. NTRU Prime. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. 3, 4, 6, 17

BCDR17. Dominic Bucerzan, Pierre-Louis Cayrel, Vlad Dragoi, and Tania Richmond. Improved timing attacks against the secret permutation in the mceliece PKC. *Int. J. Comput. Commun. Control*, 12(1):7–25, 2017. http://univagora.ro/jour/index.php/ijccc/article/view/2780. 26

BDF+11. Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Heidelberg, December 2011. 2

BDH+19. Ciprian Baetu, F. Betül Durak, Loïs Huguenin-Dumittan, Abdullah Talayhan, and Serge Vaudenay. Misuse attacks on post-quantum cryptosystems. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 747–776. Springer, Heidelberg, May 2019. 4, 6, 7, 33, 34

BDK+21. Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel-resistant implementation of SABER. *ACM J. Emerg. Technol. Comput. Syst.*, 17(2):10:1–10:26, 2021. See also https://eprint.iacr.org/2020/733. 30

BDL01. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14(2):101–119, March 2001. 2

Bel00. Mihir Bellare, editor. *CRYPTO 2000*, volume 1880 of *LNCS*. Springer, Heidelberg, August 2000. 18, 20

BG15. Johannes Blömer and Peter Günther. Singular curve point decompression attack. In Naofumi Homma and Victor Lomné, editors, *2015 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2015, Saint Malo, France, September 13, 2015*, pages 71–84. IEEE Computer Society, 2015. 2

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 Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 61–90. Springer, Heidelberg, December 2019. 2, 24

BMM00. Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In Bellare [Bel00], pages 131–146. 2

BP18. Daniel J. Bernstein and Edoardo Persichetti. Towards KEM unification. Cryptology ePrint Archive, Report 2018/526, 2018. https://eprint.iacr.org/2018/526. 25

BV98. Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 59–71. Springer, Heidelberg, May / June 1998. 2

CDH+20. Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, Tsunekazu Saito, Takashi Yamakawa, and Keita Xagawa. NTRU. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. 3, 4, 17, 28

CEvMS15. Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Differential power analysis of a McEliece cryptosystem. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 15*, volume 9092 of *LNCS*, pages 538–556. Springer, Heidelberg, June 2015. 31

Cho16. Tung Chou. QcBits: Constant-time small-key code-based cryptography. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 280–300. Springer, Heidelberg, August 2016. 31

Cos21. Craig Costello. The case for SIKE: A decade of the supersingular isogeny problem. 2021. https://eprint.iacr.org/2021/543. 4, 35

CS03. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003. 1

CT16. Jung Hee Cheon and Tsuyoshi Takagi, editors. *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*. Springer, Heidelberg, December 2016. 19

DDS+19. Jintai Ding, Joshua Deaton, Kurt Schmidt, Vishakha, and Zheng Zhang. A simple and practical key reuse attack on NTRU cryptosystem. Cryptology ePrint Archive, Report 2019/1022, 2019. https://eprint.iacr.org/2019/1022. 4, 29, 35

Den03. Alexander W. Dent. A designer's guide to KEMs. In Kenneth G. Paterson, editor, *9th IMA International Conference on Cryptography and Coding*, volume 2898 of *LNCS*, pages 133–151. Springer, Heidelberg, December 2003. 1, 25

DJP14. Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014. 2, 35

DK20. Julian Danner and Martin Kreuzer. A fault attack on the Niederreiter cryptosystem using binary irreducible Goppa codes. *Journal of Groups, Complexity, Cryptology*, 12(1), March 2020. https://gcc.episciences.org/6212. 27

DKR+20. Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. 3, 4, 17, 29

EOS07. Daniela Engelbert, Raphael Overbeck, and Arthur Schmidt. A summary of mceliece-type cryptosystems and their security. *J. Math. Cryptol.*, 1(2):151–199, 2007. 26

FH17. Wieland Fischer and Naofumi Homma, editors. *CHES 2017*, volume 10529 of *LNCS*. Springer, Heidelberg, September 2017. 20, 21

Flu16. Scott Fluhrer. Cryptanalysis of ring-LWE based key exchange with key share reuse. Cryptology ePrint Archive, Report 2016/085, 2016. https://eprint.iacr.org/2016/085. 2

FO99. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Heidelberg, August 1999. 1

FO13. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 26(1):80–101, January 2013. 1

GJN20. Qian Guo, Thomas Johansson, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 359–386. Springer, Heidelberg, August 2020. 4, 9, 17, 33

GJS16. Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. In Cheon and Takagi [CT16], pages 789–815. 3, 4, 31, 32

gm20. goulov and mandlebro. Writeup: sidhe – PlaidCTF 2020, April 2020. https://sectt.github.io/writeups/Plaid20/crypto_sidhe/README. 36

GMP21. Paul Grubbs, Varun Maram, and Kenneth G. Paterson. Anonymous, robust post-quantum public key encryption. Cryptology ePrint Archive, Report 2021/708, 2021. https://eprint.iacr.org/2021/708. 23

GN07. Nicolas Gama and Phong Q. Nguyen. New chosen-ciphertext attacks on NTRU. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 89–106. Springer, Heidelberg, April 2007. 28

GPST16. Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. On the security of supersingular isogeny cryptosystems. In Cheon and Takagi [CT16], pages 63–91. 2, 4, 35, 36

Gun19. Stjin Gunter. Timing attacks and the NTRU public-key cryptosystem, July 2019. Bachelor Thesis. https://research.tue.nl/en/studentTheses/timing-attacks-and-the-ntru-public-key-cryptosystem. 29

GW17. Alexandre Gélin and Benjamin Wesolowski. Loop-abort faults on supersingular isogeny cryptosystems. In Lange and Takagi [LT17], pages 93–106. 35

HGS99. Chris Hall, Ian Goldberg, and Bruce Schneier. Reaction attacks against several public-key cryptosystems. In Vijay Varadharajan and Yi Mu, editors, *ICICS 99*, volume 1726 of *LNCS*, pages 2–12. Springer, Heidelberg, November 1999. 2, 26

HHHK03. Daewan Han, Jin Hong, Jae Woo Han, and Daesung Kwon. Key recovery attacks on NTRU without ciphertext validation routine. In Reihaneh Safavi-Naini and Jennifer Seberry, editors, *ACISP 03*, volume 2727 of *LNCS*, pages 274–284. Springer, Heidelberg, July 2003. 28

HHK17. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Heidelberg, November 2017. 2, 6, 24

HMP10. Stefan Heyse, Amir Moradi, and Christof Paar. Practical power analysis attacks on software implementations of McEliece. In Sendrier [Sen10], pages 108–125. 26

HNP+03. Nick Howgrave-Graham, Phong Q. Nguyen, David Pointcheval, John Proos, Joseph H. Silverman, Ari Singer, and William Whyte. The impact of decryption failures on the security of NTRU encryption. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 226–246. Springer, Heidelberg, August 2003. 28, 29

HPA21. James Howe, Thomas Prest, and Daniel Apon. SoK: How (not) to design and implement post-quantum cryptography. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 444–477. Springer, Heidelberg, May 2021. 4

HPS98. Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *LNCS*, pages 267–288. Springer, Heidelberg, June 1998. 28

HRSS17. Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. In Fischer and Homma [FH17], pages 232–252. 28

HS00. Jeffrey Hoffstein and Joseph H. Silverman. Reaction attacks against the NTRU public key cryptosystem. Ntru tech report, 2000. #015v2. Available at https://ntru.org/resources.shtml. 28

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, *ACNS 20, Part I*, volume 12146 of *LNCS*, pages 208–227. Springer, Heidelberg, October 2020. 3, 4, 6, 7, 27, 28, 30, 31, 33, 34

ISO06. ISO/IEC 18033-2:2006 information technology — security techniques — encryption algorithms — part 2: Asymmetric ciphers, 2006. https://www.iso.org/standard/37971.html. 1

JAC+20. David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson. SIKE. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. 3, 4, 17, 35

JD11. David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 19–34. Springer, Heidelberg, November / December 2011. 2, 35

JJ00. Éliane Jaulmes and Antoine Joux. A chosen-ciphertext attack against NTRU. In Bellare [Bel00], pages 20–35. 28, 35

JZC+18. Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 96–125. Springer, Heidelberg, August 2018. 2, 24

JZM19. Haodong Jiang, Zhenfeng Zhang, and Zhi Ma. Key encapsulation mechanism with explicit rejection in the quantum random oracle model. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 618–645. Springer, Heidelberg, April 2019. 2, 24

KAJ17. Brian Koziel, Reza Azarderakhsh, and David Jao. Side-channel attacks on quantum-resistant supersingular isogeny Diffie-Hellman. In Carlisle Adams and Jan Camenisch, editors, *SAC 2017*, volume 10719 of *LNCS*, pages 64–81. Springer, Heidelberg, August 2017. 35

Koc96. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Heidelberg, August 1996. 2

KRSS. Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. Pqm4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4. 3, 9

KSS+20. Veronika Kuchta, Amin Sakzad, Damien Stehlé, Ron Steinfeld, and Shifeng Sun. Measure-rewind-measure: Tighter quantum random oracle model proofs for one-way to hiding and CCA security. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 703–728. Springer, Heidelberg, May 2020. 2, 6, 24

KY11. Abdel Alim Kamal and Amr M. Youssef. Fault analysis of the NTRUEncrypt cryptosystem. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 94-A(4):1156–1158, 2011. 29

KY12. Abdel Alim Kamal and Amr M. Youssef. A scan-based side channel attack on the NTRUEncrypt cryptosystem. In *2012 Seventh International Conference on Availability, Reliability and Security – ARES 2012*, pages 402–409, 2012. 29

LP11. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, Heidelberg, February 2011. 6

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010. 6

LSCH10. Mun-Kyu Lee, Jeong Eun Song, Dooho Choi, and Dong-Guk Han. Countermeasures against power analysis attacks for the NTRU public key cryptosystem. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 93-A(1):153–163, 2010. 29

LT17. Tanja Lange and Tsuyoshi Takagi, editors. *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*. Springer, Heidelberg, 2017. 19, 22

Mas69. J. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, 1969. 26

McE78. Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. The deep space network progress report 42-44, Jet Propulsion Laboratory, California Institute of Technology, January/February 1978. https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF. 2, 26

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. 3, 31

MY08. Petros Mol and Moti Yung. Recovering NTRU secret key from inversion oracles. In Ronald Cramer, editor, *PKC 2008*, volume 4939 of *LNCS*, pages 18–36. Springer, Heidelberg, March 2008. 28

NAB+20. Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. 3, 4, 17, 32, 33

NDGJ21. Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked IND-CCA secure Saber KEM. 2021. https://eprint.iacr.org/2021/079. 30

Nie86. H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):159–166, 1986. 2, 26

OP01. Tatsuaki Okamoto and David Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 159–175. Springer, Heidelberg, April 2001. 5

OUKT21. Yuki Osumi, Shusaku Uemura, Momonari Kudo, and Tsuyoshi Takagi. Key mismatch attack on SABER. In *SCIS 2021*, January 2021. In Japanese. 30

Pat75. N. Patterson. The algebraic decoding of goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975. 26

PP21. Peter Pessl and Lukas Prokop. Fault attacks on CCA-secure lattice KEMs. *IACR TCHES*, 2021(2):37–60, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8787. 2, 17

PRD+16. Martin Petrvalsky, Tania Richmond, Milos Drutarovsky, Pierre-Louis Cayrel, and Viktor Fischer. Differential power analysis attack on the secure bit permutation in the mceliece cryptosystem. In *2016 26th International Conference Radioelektronika (RADIOELEKTRONIKA)*, pages 132–137, 2016. 26

PT18. Thales Bandiera Paiva and Routo Terada. Improving the efficiency of a reaction attack on the QC-MDPC mceliece. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 101-A(10):1676–1686, 2018. 32

PT19. Thales Bandiera Paiva and Routo Terada. A timing attack on the HQC encryption scheme. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 551–573. Springer, Heidelberg, August 2019. 33

PV17. Kenneth G. Paterson and Ricardo Villanueva-Polanco. Cold boot attacks on NTRU. In Arpita Patra and Nigel P. Smart, editors, *INDOCRYPT 2017*, volume 10698 of *LNCS*, pages 107–125. Springer, Heidelberg, December 2017. 29

QCD19. Yue Qin, Chi Cheng, and Jintai Ding. An efficient key mismatch attack on the NIST second round candidate Kyber. Cryptology ePrint Archive, Report 2019/1343, 2019. https://eprint.iacr.org/2019/1343. 4, 27

QCZ+21. Yue Qin, Chi Cheng, Xiaohan Zhang, Yanbin Pan, Lei Hu, and Jintai Ding. A systematic approach and analysis of key mismatch attacks on CPA-secure lattice-based NIST candidate KEMs. Cryptology ePrint Archive, Report 2021/123, 2021. https://eprint.iacr.org/2021/123. 4, 27, 29, 33

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. 2021. https://eprint.iacr.org/2021/718. 4, 35

RHHM17. Melissa Rossi, Mike Hamburg, Michael Hutter, and Mark E. Marson. A side-channel assisted cryptanalytic attack against QcBits. In Fischer and Homma [FH17], pages 3–23. 31

RR21. Prasanna Ravi and Sujoy Sinha Roy. Side-channel analysis of lattice-based PQC candidates. NIST PQC Round 3 Seminars, 2021. https://csrc.nist.gov/projects/post-quantum-cryptography/workshops-and-timeline/round-3-seminars. 4

RRCB20. Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR TCHES*, 2020(3):307–335, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8592. 4, 27, 30, 33

RS92. Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 433–444. Springer, Heidelberg, August 1992. 1

SAB+20. Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. 3, 4, 17, 27

Sch18. John M. Schanck. A comparison of NTRU variants. Cryptology ePrint Archive, Report 2018/1174, 2018. https://eprint.iacr.org/2018/1174. 25

Sen10. Nicolas Sendrier, editor. *The Third International Workshop on Post-Quantum Cryptography, PQCRYPTO 2010*. Springer, Heidelberg, May 2010. 20, 22

Sho94. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994. 1

Sho00. Victor Shoup. Using hash functions as a hedge against chosen ciphertext attack. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 275–288. Springer, Heidelberg, May 2000. 1

Str10. Falko Strenzke. A timing attack against the secret permutation in the McEliece PKC. In Sendrier [Sen10], pages 95–107. 26

Str13. Falko Strenzke. Timing attacks against the syndrome inversion in code-based cryptosystems. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*, pages 217–230. Springer, Heidelberg, June 2013. 26

SW07. Joseph H. Silverman and William Whyte. Timing attacks on NTRUEncrypt via variation in the number of hash calls. In Masayuki Abe, editor, *CT-RSA 2007*, volume 4377 of *LNCS*, pages 208–224. Springer, Heidelberg, February 2007. 29

SXY18. Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 520–551. Springer, Heidelberg, April / May 2018. 2, 25

Ti17. Yan Bo Ti. Fault attack on supersingular isogeny cryptosystems. In Lange and Takagi [LT17], pages 107–122. 35

TT19. Akira Takahashi and Mehdi Tibouchi. Degenerate fault attacks on elliptic curve parameters in openssl. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*, pages 371–386. IEEE, 2019. 2

TU16. Ehsan Ebrahimi Targhi and Dominique Unruh. Post-quantum security of the Fujisaki-Okamoto and OAEP transforms. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 192–216. Springer, Heidelberg, October / November 2016. 2, 24

VV20. Jan Vacek and Jan Václavek. Key mismatch attack on ThreeBears, frodo and Round5. In Deukjo Hong, editor, *ICISC 20*, volume 12593 of *LNCS*, pages 182–198. Springer, Heidelberg, December 2020. 4, 33

WTBB+20. Guillaume Wafo-Tapa, Slim Bettaieb, Loïc Bidoux, Philippe Gaborit, and Etienne Marcatel. A practicable timing attack against HQC and its countermeasure. *Advances in Mathematics of Communications*, 2020. Online First. Available at https://www.aimsciences.org/article/doi/10.3934/amc.2020126. See also https://eprint.iacr.org/2019/909. 33

YJ00. Sung-Ming Yen and M. Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000. 2

YKLM02. Sung-Ming Yen, Seungjoo Kim, Seongan Lim, and Sang-Jae Moon. A countermeasure against one physical cryptanalysis may benefit another attack. In Kwangjo Kim, editor, *ICISC 01*, volume 2288 of *LNCS*, pages 414–427. Springer, Heidelberg, December 2002. 2

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. 33

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. Cryptology ePrint Archive, Report 2021/168, 2021. https://eprint.iacr.org/2021/168. 4, 29, 35

ZWW13. Xuexin Zheng, An Wang, and Wei Wei. First-order collision attack on protected NTRU cryptosystem. *Microprocessors and Microsystems*, 37(6–7):601–609, 2013. 29

$$\begin{array}{l}
\underline{\mathsf{Expt}^{\text{ow-cpa}}_{\mathsf{PKE},\mathcal{D}_{\mathcal{M}},\mathcal{A}}(\lambda)} \\
(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda) \\
m^* \leftarrow \mathcal{D}_{\mathcal{M}} \\
ct^* \leftarrow \mathsf{Enc}(pk, m^*) \\
m' \leftarrow \mathcal{A}(pk, ct^*) \\
\textbf{return } \mathsf{boole}(m' \stackrel{?}{=} \mathsf{Dec}(sk, ct^*))
\end{array}
\qquad
\begin{array}{l}
\underline{\mathsf{Expt}^{\text{ind-cpa}}_{\mathsf{PKE},\mathcal{A}}(\lambda)} \\
b \leftarrow \{0,1\} \\
(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda) \\
(m_0, m_1, st) \leftarrow \mathcal{A}_1(pk) \\
ct^* \leftarrow \mathsf{Enc}(pk, m_b) \\
b' \leftarrow \mathcal{A}_2(ct^*, st) \\
\textbf{return } \mathsf{boole}(b' \stackrel{?}{=} b)
\end{array}$$

Fig. 11: Games for PKE schemes

Supplementary Materials

A Missing Definitions

Notation: For a statement P (e.g., $r \in [0, 1]$), we define $\mathsf{boole}(P) = 1$ if P is satisfied and 0 otherwise.

For a positive integer q, we define $r' := r \bmod^+ q$ to be the unique element $r' \in [0, q)$ with $r' \equiv r \pmod q$. For an even positive integer q, we define $r' := r \bmod^\pm_\uparrow q$ (and $r' := r \bmod^\pm_\downarrow q$, resp.) to be the unique element $r' \in (-q/2, q/2]$ (and $r' \in [-q/2, q/2)$, resp.) with $r' \equiv r \pmod q$. For an element $x \in \mathbb{R}$, we define $\lceil x \rfloor = \lceil x - 1/2 \rceil \in \mathbb{Z}$, which is the nearest integer.

Security Notions of PKE: We define onewayness under chosen-plaintext attacks (OW-CPAsecurity) and indistinguishability under chosen-plaintext attacks (IND-CPAsecurity) for a PKE.

Definition A.1 (Security notions for PKE). *Let $\mathcal{D}_{\mathcal{M}}$ be a distribution over the message space $\mathcal{M}$. For any adversary $\mathcal{A}$, we define its* OW-CPA *and* IND-CPA *advantages against a PKE scheme* $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *as follows:*

$$\mathsf{Adv}^{\text{ow-cpa}}_{\mathsf{PKE},\mathcal{D}_{\mathcal{M}},\mathcal{A}}(\lambda) := \Pr[\mathsf{Expt}^{\text{ow-cpa}}_{\mathsf{PKE},\mathcal{D}_{\mathcal{M}},\mathcal{A}}(\lambda) = 1],$$
$$\mathsf{Adv}^{\text{ind-cpa}}_{\mathsf{PKE},\mathcal{A}}(\lambda) := \left|\Pr[\mathsf{Expt}^{\text{ind-cpa}}_{\mathsf{PKE},\mathcal{A}}(\lambda) = 1] - 1/2\right|,$$

where $\mathsf{Expt}^{\text{ow-cpa}}_{\mathsf{PKE},\mathcal{D}_{\mathcal{M}},\mathcal{A}}(\lambda)$ and $\mathsf{Expt}^{\text{ind-cpa}}_{\mathsf{PKE},\mathcal{A}}(\lambda)$ are experiments described in Figure 11. We say that PKE *is* OW-CPA-*secure (and* IND-CPA-*secure resp.) if $\mathsf{Adv}^{\text{ow-cpa}}_{\mathsf{PKE},\mathcal{D}_{\mathcal{M}},\mathcal{A}}(\lambda)$ (and $\mathsf{Adv}^{\text{ind-cpa}}_{\mathsf{PKE},\mathcal{A}}(\lambda)$) is negligible for any PPT adversary $\mathcal{A}$. We omit $\mathcal{D}_{\mathcal{M}}$ from* OW-CPA *security if $\mathcal{D}_{\mathcal{M}}$ is the uniform distribution over $\mathcal{M}$.*

Security Notions of KEM: We define indistinguishability under chosen-ciphertext attacks (IND-CCA security) for KEM.

Definition A.2. *For any adversary $\mathcal{A}$, we define its* IND-CCA *advantage against a KEM scheme* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *as follows:*

$$\mathsf{Adv}^{\text{ind-cca}}_{\mathsf{KEM},\mathcal{A}}(\lambda) := \left|\Pr[\mathsf{Expt}^{\text{ind-cca}}_{\mathsf{KEM},\mathcal{A}}(\lambda) = 1] - 1/2\right|,$$

where $\mathsf{Expt}^{\text{ind-cca}}_{\mathsf{KEM},\mathcal{A}}(\lambda)$ is an experiment described in Figure 12.

We say that KEM *is* IND-CCA-*secure if $\mathsf{Adv}^{\text{ind-cca}}_{\mathsf{KEM},\mathcal{A}}(\lambda)$ is negligible for any PPT adversary $\mathcal{A}$.*

B The variants of FO: $\mathsf{FO}^{\not\perp\prime}$, $\mathsf{HFO}^{\perp}$, $\mathsf{HFO}^{\not\perp}$, SXY, and SXY-KC

B.1 Another FO with implicit rejection, $\mathsf{FO}^{\not\perp\prime}$

$\mathsf{FO}^{\not\perp\prime}$ is a slightly modified version of $\mathsf{FO}^{\not\perp}$, which is used by Kyber, Saber, and FrodoKEM. The difference from $\mathsf{FO}^{\not\perp}$ is how to generate K in Encaps and Decaps.

Let $\{0,1\}^{\ell(\lambda)}$ be the plaintext space of PKE. Let $\mathsf{G}\colon \{0,1\}^* \to \mathcal{R}_{\mathsf{Enc}}$, $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^{\ell(\lambda)}$, and $\mathsf{KDF}\colon \{0,1\}^{\ell(\lambda)} \times \{0,1\}^{\ell(\lambda)} \to \{0,1\}^{k(\lambda)}$ be hash functions modeled by the random oracles. The $\mathsf{FO}^{\not\perp\prime}$ is summarized as Figure 13. One might consider assuming the IND-CPA security of PKE, the obtained KEM scheme is IND-CCA-secure in the QROM. However, this very subtle change causes some technical barrier to apply existing security proofs. See the details for Grubbs, Maram, and Paterson [GMP21].

$\mathsf{Expt}^{\text{ind-cca}}_{\mathsf{KEM},\mathcal{A}}(\lambda)$	$\textsc{Dec}_{ct^*}(ct)$
$b \leftarrow \{0,1\}$	if $ct = ct^*$, return $\perp$
$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$	$K := \mathsf{Decaps}(sk, ct)$
$(ct^*, K_0^*) \leftarrow \mathsf{Encaps}(pk)$	**return** K
$K_1^* \leftarrow \mathcal{K}$	
$b' \leftarrow \mathcal{A}^{\textsc{Dec}_{ct^*}(\cdot)}(pk, ct^*, K_b^*)$	
return $\mathrm{boole}(b' \stackrel{?}{=} b)$	

Fig. 12: Games for KEM schemes

$\mathsf{Gen}(1^\lambda)$	$\mathsf{Encaps}(pk)$	$\mathsf{Decaps}(\overline{sk}, ct)$, where $\overline{sk} = (sk, pk, h, s)$
$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$	$m \leftarrow \{0,1\}^{\ell(\lambda)}$	$m' := \mathsf{Dec}(sk, ct)$
$h \leftarrow \mathsf{H}(pk)$	$m \leftarrow \mathsf{H}(m)$ // for Kyber and Saber	$(\bar{K}', r') := \mathsf{G}(m', h)$
$s \leftarrow \{0,1\}^{\ell(\lambda)}$	$(\bar{K}, r) \leftarrow \mathsf{G}(m, \mathsf{H}(pk))$	$ct' := \mathsf{Enc}(pk, m'; r')$
$\overline{sk} \leftarrow (sk, pk, h, s)$	$ct := \mathsf{Enc}(pk, m; r)$	**if** $ct = ct'$, **then return** $K := \mathsf{KDF}(\bar{K}', \mathsf{H}(ct))$
return $(pk, \overline{sk})$	$K := \mathsf{KDF}(\bar{K}, \mathsf{H}(ct))$	**else return** $K := \mathsf{KDF}(s, \mathsf{H}(ct))$
	return (K, ct)	

Fig. 13: $\mathsf{KEM} := \mathsf{FO}^{\not\perp\prime}[\mathsf{PKE}, \mathsf{G}, \mathsf{H}, \mathsf{KDF}]$ in Kyber, Saber, and FrodoKEM.

B.2 FO with additional hash, $\mathsf{HFO}^{\perp}$ and $\mathsf{HFO}^{\not\perp}$

$\mathsf{HFO}^{\perp}$ and $\mathsf{HFO}^{\not\perp}$ (as known as $\mathsf{QFO}^{\perp}$ and $\mathsf{QFO}^{\not\perp}$) [TU16, HHK17, JZC+18, JZM19] transform a weakly-secure probabilistic PKE into IND-CCA-secure KEM like FO and add hash value of the message. Those variants are used by HQC and ntrulpr of NTRU Prime, respectively.

Let $\{0,1\}^{\ell(\lambda)}$ be the plaintext space of PKE. Let $\mathsf{G}\colon \{0,1\}^* \to \mathcal{R}_{\mathsf{Enc}}$, $\mathsf{F}\colon \{0,1\}^{\ell(\lambda)} \times \{0,1\}^* \to \{0,1\}^{\ell'(\lambda)}$, and $\mathsf{KDF}\colon \{0,1\}^{\ell(\lambda)} \times (\mathcal{C}_{\mathsf{PKE}} \times \{0,1\}^{\ell'(\lambda)}) \to \{0,1\}^{k(\lambda)}$ be hash functions modeled by the random oracles. The $\mathsf{HFO}^{\perp}$ and $\mathsf{HFO}^{\not\perp}$ is summarized as Figure 14. Assuming the IND-CPA security of PKE, the obtained KEM scheme is IND-CCA-secure in the QROM. See e.g., [KSS+20]. For the case of explicit rejection $\mathsf{HFO}^{\perp}$, we need to invoke [BHH+19, Theorem 4].

$\mathsf{Gen}(1^\lambda)$	$\mathsf{Encaps}(pk)$	$\mathsf{Decaps}(\overline{sk}, ct)$, where $\overline{sk} = (sk, pk[, s])$
$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$	$m \leftarrow \{0,1\}^{\ell(\lambda)}$	$m' := \mathsf{Dec}(sk, ct)$
$s \leftarrow \{0,1\}^{\ell(\lambda)}$ // for ntrulpr	$r \leftarrow \mathsf{G}(m)$	$r' \leftarrow \mathsf{G}(m')$
$\overline{sk} \leftarrow (sk, pk, s)$ // for ntrulpr	$ct_0 := \mathsf{Enc}(pk, m; r)$	$ct_0' := \mathsf{Enc}(pk, m'; r')$
$\overline{sk} \leftarrow (sk, pk)$ // for HQC	$ct_1 := \mathsf{F}(m, pk)$	$ct_1' := \mathsf{F}(m', pk)$
return $(pk, \overline{sk})$	$ct := (ct_0, ct_1)$	$ct' := (ct_0', ct_1')$
	$K := \mathsf{KDF}(m, ct)$	**if** $ct = ct'$, **then return** $K := \mathsf{KDF}(m', ct)$
	return (K, ct)	**else return** $K := \perp$ // for HQC
		else return $K := \mathsf{KDF}(s, ct)$ // for ntrulpr

Fig. 14: $\mathsf{KEM} := \mathsf{HFO}^{\not\perp}[\mathsf{PKE}, \mathsf{G}, \mathsf{F}, \mathsf{KDF}]$ for ntrulpr of NTRU Prime and $\mathsf{HFO}^{\perp}[\mathsf{PKE}, \mathsf{G}, \mathsf{F}, \mathsf{KDF}]$ for HQC.

B.3 SXY

SXY transforms a weakly-secure *deterministic* PKE into IND-CCA-secure KEM. This variant is employed by NTRU (ntruhps and ntruhrss).

Let $\mathcal{M}$ be the plaintext space of PKE. Let $\mathsf{KDF}\colon \mathcal{M} \to \{0,1\}^{k(\lambda)}$ and $\mathsf{H}_0\colon \{0,1\}^{\ell(\lambda)} \times C_{\mathsf{PKE}} \to \{0,1\}^{k(\lambda)}$ be hash functions modeled by the random oracles. The SXY is summarized as Figure 15. Assuming 'disjoint-simulatability'[20] of PKE, the obtained KEM scheme is IND-CCA-secure in the QROM [SXY18].

$\mathsf{Gen}(1^\lambda)$	$\mathsf{Encaps}(pk)$	$\mathsf{Decaps}(\overline{sk}, ct)$, where $\overline{sk} = (sk, pk, s)$
$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$	$m \leftarrow \mathcal{M}$	$m' := \mathsf{Dec}(sk, ct)$
$s \leftarrow \{0,1\}^l$	$ct := \mathsf{Enc}(pk, m)$	**if** $m' = \bot$, **then return** $K := \mathsf{H}_0(s, ct)$
$\overline{sk} \leftarrow (sk, pk, s)$	$K := \mathsf{KDF}(m)$	$ct' := \mathsf{Enc}(pk, m')$
return $(pk, \overline{sk})$	**return** (K, ct)	**if** $ct = ct'$, **then return** $K := \mathsf{KDF}(m')$
		else return $K := \mathsf{H}_0(s, ct)$

Fig. 15: $\mathsf{KEM} := \mathsf{SXY}[\mathsf{PKE}, \mathsf{KDF}, \mathsf{H}_0]$ in NTRU (ntruhps and ntruhrss).

Remark B.1. NTRU also omits an explicit re-encryption check invoking Enc. In NTRU, a ciphertext is $c = hr + m$, where h is a public key and $(r, s) \in \mathcal{R} \times \mathcal{M}$ is a plaintext. The decryption algorithm first computes $r' \in rs$, computes $m' = c - hr'$, and checks if (r', m') is in $\mathcal{R} \times \mathcal{M}$ or not. This check is equivalent to checking $c = hr' + m'$, because h is *invertible*. See [Sch18, Section 5.1] for the details.

Bernstein and Persichetti dubbed this property *rigidity* [BP18].

B.4 SXY with additional hash, SXY-KC

The final one is a transformation that transforms a weakly-secure *deterministic* PKE into IND-CCA-secure KEM, employed by Classic McEliece and sntrupr of NTRU Prime. We interpret the transformation as SXY-KC, because the security proof of Classic McEliece is inspired by Dent [Den03] and Saito et al. [SXY18].

Let $\mathcal{M}$ be the plaintext space of PKE. Let $\mathsf{F}\colon \mathcal{M} \to \{0,1\}^{\ell'(\lambda)}$. Let $\mathsf{KDF}\colon \mathcal{M} \times (C_{\mathsf{PKE}} \times \{0,1\}^{\ell'(\lambda)}) \to \{0,1\}^{k(\lambda)}$ be a hash function modeled by the random oracle. The SXY-KC is summarized as Figure 16. Assuming 'disjoint-simulatability' of PKE, the obtained KEM scheme is IND-CCA-secure in the QROM [SXY18, ABC+20].

$\mathsf{Gen}(1^\lambda)$	$\mathsf{Encaps}(pk)$	$\mathsf{Decaps}(\overline{sk}, ct)$, where $\overline{sk} = (sk, pk, z)$ and $ct = (ct_0, ct_1)$
$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$	$m \leftarrow \mathcal{M}$	$m' := \mathsf{Dec}(sk, ct_0)$
$s \leftarrow \{0,1\}^l$	$ct_0 := \mathsf{Enc}(pk, m)$	**if** $m' = \bot$, **then return** $K := \mathsf{KDF}(s, ct)$
$\overline{sk} \leftarrow (sk, pk, s)$	$ct_1 := \mathsf{F}(m)$	$ct'_0 := \mathsf{Enc}(pk, m')$
return $(pk, \overline{sk})$	$ct := (ct_0, ct_1)$	$ct'_1 := \mathsf{F}(m')$
	$K := \mathsf{KDF}(m, ct)$	$ct := (ct'_0, ct'_1)$
	return (K, ct)	**if** $ct = ct'$, **then return** $K := \mathsf{KDF}(m', ct)$
		else return $K := \mathsf{KDF}(s, ct)$

Fig. 16: $\mathsf{KEM} := \mathsf{SXY\text{-}KC}[\mathsf{PKE}, \mathsf{F}, \mathsf{KDF}]$ in Classic McEliece and sntrupr of NTRU Prime.

Remark B.2. One might wonder Decaps in Classic McEliece has no explicit re-encryption check ([ABC+20, Sec.2.3.3]). In their specification, Dec in Classic McEliece internally checks $ct'_0 = \mathsf{Enc}(pk, m')$ or not ([ABC+20, Sec.2.2.4]).

[20] Roughly speaking, disjoint-simulatability means that a random ciphertext is indistinguishable from a random string in C_{PKE}, that is, $\mathsf{Enc}(pk, U(\mathcal{M})) \sim_c U(C_{\mathsf{PKE}})$, and $\#(\mathsf{Enc}(pk, \mathcal{M}) \cap C_{\mathsf{PKE}}) \ll \#C_{\mathsf{PKE}}$.

C Survey of Key-Recovery Plaintext-Checking Attacks

C.1 Classic McEliece

On the McEliece and Niederreiter PKE, we recommend to read an excellent servery by Engelbert, Overbeck, and Schmidt [EOS07]. To the best of the authors' knowledge, there are no key-recovery plaintext-checking attack against the McEliece and Niederreiter PKE [McE78, Nie86].

Review of Classic McEliece: Classic McEliece [ABC+20] is based on the Niederreiter PKE, in which a public key is a scrambled parity-check matrix, a plaintext is an error vector, and a ciphertext is a syndrome.

Table 7: Parameter sets of Classic McEliece in Round 3. Note that $q = 2^m$ and $k = n - mt$. (We omit the semi-systematic forms.)

parameter sets	m	n	t	k
`kem/mceliece348864`	12	3488	64	2720
`kem/mceliece460896`	13	4608	96	3360
`kem/mceliece6688128`	13	6688	128	5024
`kem/mceliece6960119`	13	6960	119	5413
`kem/mceliece8192128`	13	8192	128	6528

Define $\mathcal{S} = \{e \in \mathbb{F}_2^n \mid \mathsf{HW}(e) = t\}$, which is a plaintext space. The underlying PKE of Classic McEliece is summarized as follows, where we only consider the systematic form and omit the details for the semi-systematic form:

- $\mathsf{Gen}(pp)$: Choose a monic irreducible polynomial g in $\mathbb{F}_q[x]$ of degree t and distinct $\alpha_1, \dots, \alpha_n \leftarrow \mathbb{F}_q$. Compute a parity-check matrix $\hat{H} \in \mathbb{F}_2^{n \times k}$ of the Goppa code generated by g and $\alpha_1, \dots, \alpha_n$. Reduce $\hat{H}$ to systematic form $[I_{n-k} \mid T]$. (If this fails, return $\perp$). Output $pk = T \in \mathbb{F}_2^{(n-k)\times k}$ and $sk = (T, \Gamma)$, where $\Gamma = (g, \alpha_1, \dots, \alpha_n)$. We note that using Γ, one can correct an error of the codeword up to t, because the minimum distance of the Goppa code is at least $2t + 1$ by design.
- $\mathsf{Enc}(pk, e)$: Define $H = [I_{n-k} \mid T] \in \mathbb{F}_2^{(n-k)\times n}$. Compute $c = He \in \mathbb{F}_2^{n-k}$. Output c.
- $\mathsf{Dec}(sk, c)$: Extend c to $v = (c, 0, \dots, 0) \in \mathbb{F}_2^n$. Find the unique codeword c in the Goppa code defined by Γ' that satisfies $\mathsf{HW}(c, v) \le t$. Set $e = v + c$. If $\mathsf{HW}(e) = t$ and $c = He$, then return e. Otherwise, return $\perp$.

Notice that there are no specification on the decoding algorithm and we can choose it from existing decoding algorithms: The submitted implementation uses the Berlekamp-Massey decoding algorithm [Mas69], which corrects the error up to t. We can use the Patterson decoding algorithm [Pat75] which sometimes corrects a $(t + 1)$-error. However, the decryption algorithm checks the Hamming weight of e and such $(t + 1)$-error will be rejected.

Review of Key-recovery SCA/FIA against the McEliece/Niederreiter PKE: Hall, Goldberg, and Schneier [HGS99] gave a message-recovery reaction attack against the McEliece PKE. The attack queries a ciphertext plus an i-th unit vector and determines the i-th bit of error by seeing the decryption error occurs or not (this corresponding to $t{+}1$ error or $t{-}1$ error). They also pointed out if the unpermuted code is public and the decryption oracle internally decodes $t + 1$ errors sometimes, then it could be used to determine the secret permutation matrix by calculating decryption failure rate on each bits. Engelbert, Overbeck, and Schmidt [EOS07] surveyed the security of the McEliece and Niederreiter PKEs. They pointed out that, when the support of the Goppa code is known, if one can obtain the permutation matrix, then one can recover the whole secret key. Strenzke [Str10] gave a timing attack against the McEliece PKE using Patterson's decoding algorithm which retrieves the secret permutation. Heyse, Moradi, and Paar [HMP10] gave key-recovery side-channel attacks against the McEliece PKE using Patterson's decoding algorithm based on power consumption. Strenzke [Str13] improve his attack and gave practical key-recovery timing attack which exploits the secret permutation of the McEliece PKE using Patterson's decoding algorithm. Petrvalsky, Richmond, Drutarovsky, Cayrel, and Fischer [PRD+16] gave a DPA-based side-channel attack against the McEliece PKE with the Patterson decoder which retrieves a secret permutation. Bucerzan, Cayrel, Dragoi, and Richmond [BCDR17] improved timing attack against the McEliece PKE with the Patterson decoder. Danner and Kreuzer [DK20] gave a key-recovery fault-injection attack against the Niederreiter PKE with a binary irreducible Goppa code. As we saw, there are no known key-recovery attacks using the plaintext-checking oracle.

C.2 Kyber

Review of Kyber in Round 3: Kyber [SAB+20] is a KEM scheme based on the Module LWE problem. We briefly review the underlying PKE scheme of Kyber.

Table 8: Parameter sets of Kyber in Round 3.

parameter sets	n	k	q	η_1	η_2	d_U	d_V
Kyber512	256	2	3329	3	2	10	4
Kyber768	256	3	3329	2	2	10	4
Kyber1024	256	4	3329	2	2	11	5

Define $\mathcal{R} = \mathbb{Z}[x]/(x^n + 1)$ and $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$. For a positive integer η, we define a central-binomial distribution Ψ_η as $(a_1, b_1, \dots, a_\eta, b_\eta) \leftarrow \{0,1\}^{2\eta}$ and return $\sum_{i=1}^{\eta}(a_i - b_i)$. For a polynomial $P \in \mathcal{R}$, $P \leftarrow \Psi_\eta$ implies each coefficient of the polynomial is chosen from Ψ_η independently.

For $x \in \mathbb{Z}$, we define two functions: $\mathsf{comp}_q(x, d) := \lceil (2^d/q) \cdot x \rfloor \bmod^{\pm} 2^d$ and $\mathsf{decomp}_q(x, d) := \lceil (q/2^d) \cdot x \rfloor$. For $x = (x_1, \dots, x_k) \in \mathbb{Z}^k$ with some k, we define $\mathsf{comp}_q(x, d) = (\mathsf{comp}_q(x_1, d), \dots, \mathsf{comp}_q(x_k, d))$ and $\mathsf{decomp}_q(x, d) = (\mathsf{decomp}_q(x_1, d), \dots, \mathsf{decomp}_q(x_k, d))$

We have $\left|(x - \mathsf{decomp}_q(\mathsf{comp}_q(x, d), d)) \bmod^{\pm} q\right| \le \lceil q/2^{d+1} \rfloor$. We also note that $\mathsf{comp}_q(x, 1) = 1$ if $|x \bmod^{\pm} q| \le \lceil q/4 \rfloor$ and 0 otherwise.

The underlying PKE scheme of Kyber is summarized as follows:

- $\mathsf{Gen}(pp)$: $A \leftarrow \mathcal{R}_q^{k\times k}$ and $(sk, d) \leftarrow (\Psi_{\eta_1}^k)^2$. Output $pk = (A, B) = (A, A \cdot sk + d)$ and sk.
- $\mathsf{Enc}(pk, \mu)$: Sample $t \leftarrow \Psi_{\eta_1}^k$, $e \leftarrow \Psi_{\eta_2}^k$, and $f \leftarrow \Psi_{\eta_2}$. Compute $(U, V) = (tA+e, tB+f+\lceil q/2 \rfloor \cdot \mu) \in \mathcal{R}_q^k \times \mathcal{R}_q$. Output $(U', V') = (\mathsf{comp}_q(U, d_U), \mathsf{comp}_q(V, d_V))$.
- $\mathsf{Dec}(sk, (U', V'))$: Compute $(U, V) = (\mathsf{decomp}_q(U', d_U), \mathsf{decomp}_q(V', d_V))$. Return $\mu' = \mathsf{comp}_q(V - U \cdot sk, 1)$.

Review of key-recovery attacks against Kyber: There are key-recovery attacks against Kyber exploiting a plaintext-checking oracle or key-mismatch oracle: Qin, Cheng, and Ding [QCD19], Ravi, Roy, Chattopadhyay, and Bhasin [RRCB20], Huguenin-Dumittan and Vaudenay [HV20], and Qin, Cheng, Zhang, Pan, Hu, and Ding [QCZ+21].

We follow the KR-PCA against Kyber512 in Huguenin-Dumittan and Vaudenay [HV20]. Kyber has three parameter sets Kyber512, Kyber756, and Kyber1024. In Round 3, the submitter changed some parameters for Kyber512 from those of Round 2.[21] Since Huguenin-Dumittan and Vaudenay [HV20] (and Qin et al. [QCD19]) targets the Round 2 version of Kyber512, we need to adjust parameters in the attack.

Key-recovery attack against Kyber512 in Round 3: Let $\mu = 0 \in \mathcal{R}$ and let $\rho = \lceil q/4 \rfloor = 832$. Suppose that we query (U', V') to the plaintext-checking oracle with candidate plaintext $\mu = 0$. For ease of notation, we define $\delta = (\delta_0, \dots, \delta_{n-1}) := V - U \cdot sk - \mathsf{encode}(pt)$, where $U = \mathsf{decomp}_q(U', d_U)$ and $V = \mathsf{decomp}_q(V', d_V)$. The plaintext-checking oracle returns T if and only if $|\delta_i \bmod^{\pm} q| \le \rho$ for all $i \in [n]$. (If so, (U', V') is decrypted into 0.)

Lemma C.1. *Let $U = (-276, 0)$ and $U' = \mathsf{comp}_q(U, d_U)$. Let $t \in \{-5, -4, -3, \dots, 4, 5\}$. Let $V = 208t \cdot x^i = (0, \dots, 208 \cdot t, \dots, 0)$ be a polynomial with $208t$ in the i-th coefficient and 0 elsewhere and $V' = \mathsf{comp}_q(V, d_V)$. Let $m' = \mathsf{Dec}(sk, (U', V'))$. We have $m'_j = 0$ for all j but i and*

$$m'_i = 0 \iff |276 \cdot sk_i + 208 \cdot t| \le \rho.$$

[21] See "Increase noise parameter for Kyber512" and "Reduce ciphertext compression of Kyber512" in Changes to the core Kyber design. Kyber512 has parameters $(n, q, \eta_1, \eta_2, d_U, d_V) = (256, 3329, 2, 2, 10, 3)$ in Round 2 and $= (256, 3329, 3, 2, 10, 4)$ in Round 3

Table 9: The behavior of m'_i of $m' = \mathrm{Dec}(sk, (U', V'))$ on a ciphertext (U', V') with $U = (-276, 0^{k-1})$ and $V = 208t \cdot x^i$.

(a) Kyber512

sk_i \ t	−3	−2	−1	0	1	2	3
−3	1	1	1	0	0	0	0
−2	1	1	0	0	0	0	0
−1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
+1	0	0	0	0	0	0	1
+2	0	0	0	0	0	1	1
+3	0	0	0	0	1	1	1

(b) Kyber768 and Kyber1024

sk_i \ t	−3	−2	−1	0	1	2	3
−2	1	1	0	0	0	0	0
−1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
+1	0	0	0	0	0	0	1
+2	0	0	0	0	0	1	1

The proof is very similar to that of Huguenin-Dumittan and Vaudenay [HV20, Lemma 1] and we omit it.

See the pattern in Table 9a for $sk_i = s$ and $t \in \{-5, -4, \dots, 4, 5\}$. According to the table, we examine $t \in \{-3, -2, -1, 1, 2, 3\}$ to determine $sk_i \in \{-3, -2, \dots, 2, 3\}$.

Trade-off: In the case of the skipping-equality-test attack, we can reduce the number of queries as in the case of ntrulpr. For example, we can determine $sk_0, \dots, sk_{\ell-1}$ by six faulty decapsulation results as follows:

1. For $t \in \{-3, -2, -1, 1, 2, 3\}$, we prepare ciphertext (U', V'_t) with $U = (-276, 0)$ and $V_t = \sum_{i=0}^{\ell-1} 208tx^i$ and obtain faulty decapsulation results $K'_t = \mathsf{KDF}(m'_t, (U', V'_t))$ where $m'_t = \mathrm{Dec}(sk, (U', V'_t))$.
2. We then compute $K_{\mathrm{guess},t} = \mathsf{KDF}(m_{\mathrm{guess}}, (U', V'_t))$ for all $m_{\mathrm{guess}} = m'' \| 0^{256-\ell}$ with $m'' \in \{0, 1\}^\ell$ and t.
3. We determine the i-th coefficient of m'_t by comparing $K_{\mathrm{guess},t}$ with K_t.
4. We determine sk_i by using the table and the i-th coefficients of all m'_t

Key-recovery attack against Kyeber768 and Kyber1024: We examine $U = (-276, 0^{k-1})$ with $V = 208 \cdot t \cdot x^i$ and obtain the same table as Kyber512, while $sk_i \in [-2, +2]$ because $\eta_1 = 2$ in Kyber768 and Kyber1024. See Table 9b for the behavior of m'_i of $m' = \mathrm{Dec}(sk, (U', V'))$ with $U = (-276, 0^{k-1})$ and $V = 208 \cdot t \cdot x^i$ for $t \in \{-3, -2, \dots, 2, 3\}$. According to the table, we can determine dk_i by querying (U', V') for $t \in \{-3, -2, +2, +3\}$.

Again, in the case of the skipping-equality-test attack, we can reduce the number of queries as in the case of ntrulpr. We can determine ℓ coefficients of sk by four faulty decapsulation results as in the case of Kyber512.

C.3 NTRU

Review of NTRU: NTRU [CDH+20] is based on NTRU [HPS98] and NTRU-HRSS [HRSS17]. We here briefly review NTRU [HPS98].

Define $\mathcal{R} = \mathbb{Z}[x]/(x^n - 1)$ and $\mathcal{R}_a = \mathbb{Z}_a[x]/(x^n - 1)$ for $a = q = 2^{\epsilon_q}$ and 3. Let $\mathcal{T} = \{\sum_{i=0}^{n-1} t_i x^i \mid t_i \in \{-1, 0, +1\}\}$. Let $\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_r, \mathcal{L}_m \subseteq \mathcal{T}$ be carefully-chosen subsets of $\mathcal{T}$.

The underlying PKE scheme of NTRU is summarized as follows:

- $\mathrm{Gen}(pp)$: $(f, g) \leftarrow \mathcal{L}_f \times \mathcal{L}_g$. Compute $f_q = f^{-1} \in \mathcal{R}_q$ and $h = 3gf_q \in \mathcal{R}_q$. Output $pk = h$ and $sk = (f, h)$.
- $\mathrm{Enc}(pk, (r, m))$: Output $c = hr + m \in \mathcal{R}_q$.
- $\mathrm{Dec}(sk, c)$: Compute $a = cf \bmod^{\pm}_{\downarrow} q$. $m = a \cdot f^{-1} \bmod^{\pm} 3$. $r = (c - m) \cdot h^{-1} \bmod^{\pm}_{\downarrow} q$. If $(r, m) \in \mathcal{L}_r \times \mathcal{L}_m$ then return (r, m); else, return $\perp$.

Review of key-recovery attacks against NTRU: Hoffstein and Silverman [HS00], Jaulmes and Joux [JJ00], Han, Hong, Han, and Kwon [HHHK03], and Mol and Yung [MY08] gave key-recovery reaction attacks against *unpadded* NTRU. We note that the key-recovery attack of Hoffstein and Silverman [HS00] and one of key-recovery attacks in Jaulmes and Joux [JJ00] can be used in the key-recovery plaintext-checking attacks. Howgrave-Graham, Nguyen, Pointcheval, Proos, Silverman, Singer, and Whyte [HNP+03] and Gama and Nguyen [GN07] gave key-recovery chosen-ciphertext attacks against *padded* NTRUs.

Ding, Deaton, Schmidt, Vishakha, and Zhang [DDS+19] gave a KR-PCA against NTRU. Since the target of their attack [DDS+19] is old-school unpddded NTRU, we need to adjust their attack in order to make ntruhps of NTRU in Round 3 as a target but the adjustmen is subtle and we omit the detail. Zhang, Cheng, Qin, and Ding [ZCQD21] recently gave a KR-PCA against ntruhrss of NTRU.

Review of key-recovery SCA/FIA against NTRU: Moreover, there are a lot of key-recovery SCA/FIA against NTRU. Silverman and Whyte [SW07] proposed a key-recovery timing attack against padded NTRUEncrypt. Atici, Batina, Gierlichs, and Verbauwhede [ABGV08] gave key-recovery differential power analysis attack against NTRU on FPGA. Lee, Song, Choi, and Han [LSCH10] also gave key-recovery attack using the correlation power analysis against software implementaiton of NTRU. Kamal and Youssef gave key-recovery reaction attakc using fault [KY11] and scan-based key-recovery SCA [KY12]. Zheng, Wang, and Wei [ZWW13] gave another key-recovery attack using power analysis against NTRU. Paterson and Villanueva-Polanco gave a cold-boot attack against NTRU [PV17]. Gunter discussed timing attacks against reference implementations of NTRU [Gun19]. Askeland and Rønjom [AR21] proposed a key-recovery side-channel attack exploiting EM leakage of unpack of the secret key. All of them do not exploit the plaintext-checking/key-mismatch oracle.

KR-PCA against NTRU: We review how their KR-PCA [DDS+19] works against *old-school* NTRU here. We omit the details of their KR-PCAs [DDS+19, QCZ+21] against NTRU-HPS and NTRU-HRSS.

Notice that there are equivalent keys $(\hat{f}, \hat{g}) = (\pm f x^i, \pm g x^{-i})$ such that $h = 3\hat{g}\hat{f}^{-1} \bmod q$. It is enough to get one of such $\hat{g}$ to recover whole secret key $\hat{f}$ and $\hat{g}$, since we can obtain $\hat{f} = 3\hat{g} \cdot h^{-1} \bmod q$.

Determine the longest chain: We call g's sequential coefficients as *chain* if they are 1's ore -1's: A chain in g is $(s, l) \in [n]^2$ such that $g_{s \bmod n} = g_{s+1 \bmod p} = \cdots = g_{s+l-1 \bmod p} = v \in \{-1, +1\}$. The length of chain (s, l) is l. Their key-recovery attack first find the length of the longest chain of g and assume that $g_0 = g_1 = \cdots = g_{l-1} = 1$.

The attack first determine the length of the longest chain as follows: Let us consider $r = \sum_{i=0}^{j-1} t_j \cdot x^i$, where t_j is carefully chosen. We query a ciphertext $hr + 0$ and a corresponding plaintext. In the decryption, we will have $a \equiv cf \equiv pgr \pmod q$ and $a_i \equiv pt_j(g_{i \bmod n} + g_{i-1 \bmod n} + \cdots + g_{i-(j-1)} \bmod n) \pmod q$. For $j = 1, \dots, k-1$, we will get 'mismatch' because t_j is chosen as $\max\{|a_i|\} > q/2$ and, for $j = k$, we get 'match' because $\max\{|a_i|\} < q/2$. The mismatch occurs a result of *wrap failure* in attacks against NTRU [HNP+03]. (We note that we can obtain the same result by setting $c = h \cdot 0 + m$ with $m = \sum_{i=0}^{j-1} t_j \cdot x^i$. In the case, we obtain $a = mf \pmod q$.)

Determine the rest: Once we assume $\hat{g} = (1, 1, \dots, 1, ?, \dots, ?)$, we can determine rest coefficients by querying two ciphertexts for each coefficients. For simplicity, we assume that the longest chain is unique.

In order to determine j-th coefficient for $j \geq k$, we query two ciphertexts hr^+ and hr^-, where $r^\pm = \sum_{i=0}^{k-1} tx^i \pm t \cdot x^j$, with corresponding plaintexts.

Number of queries: In order to determine the length k of the longest chain, we need k faulty decapsulation results. In order to determine the rest coefficients, we need 2 faulty-decapsulation results on each coefficients. Thus, we need at most $3n$ to recover a whole key. ($2n + 10$ may be enough for 99% secret keys.)

C.4 Saber

Review of Saber: Saber [DKR+20] is a KEM scheme based on the Module LWR problem. Saber has three parameter sets LightSaber (lv.1), Saber (lv.3), and FireSaber (lv.5). See Table 10.

Table 10: Parameter sets of Saber in Round 3.

parameter sets	n	k	q	p	T	μ
`LightSaber`	256	2	8192	1024	8	10
`Saber`	256	3	8192	1024	16	8
`FireSaber`	256	4	8192	1024	64	6

Define $\mathcal{R} = \mathbb{Z}[x]/(x^n + 1)$ and $\mathcal{R}_a = \mathbb{Z}_a[x]/(x^n + 1)$ for $a = q, p, T, 2$. Let $\epsilon_q = \lg(q)$, $\epsilon_p = \lg(p)$, and $\epsilon_T = \lg(T)$. For an even positive integer μ, we define a central-binomial distribution β_η as $(a_1, b_1, \dots, a_{\mu/2}, b_{\mu_2}) \leftarrow$

Table 11: Saber and FireSaber: The behavior of m'_i of $m' = \mathrm{Dec}(sk, (U, V))$ on a ciphertext (U, V) with $U = (u, 0^{k-1})$ and $V = t \cdot x^i$.

(a) Saber with $u = 54$ and 57

sk_i \ t	$u=54$: 0	1	$u=57$: 0	1	2	3	4	5	6	7
−4	0	1	0	1	1	1	1	1	1	1
−3	0	0	0	1	1	1	1	1	1	1
−2	0	0	0	0	1	1	1	1	1	1
−1	0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	0	0	1	1	1	1
+1	0	0	0	0	0	0	0	1	1	1
+2	0	0	0	0	0	0	0	0	1	1
+3	0	0	0	0	0	0	0	0	0	1
+4	0	0	0	0	0	0	0	0	0	0

(b) FireSaber with $u = 15$

sk_i \ t	0	13	14	15	16	17	18
−3	0	1	1	1	1	1	1
−2	0	0	1	1	1	1	1
−1	0	0	0	1	1	1	1
0	0	0	0	0	1	1	1
+1	0	0	0	0	0	1	1
+2	0	0	0	0	0	0	1
+3	0	0	0	0	0	0	0

$\{0,1\}^\mu$ and return $\sum_{i=1}^{\mu/2}(a_i - b_i) \in [-\mu/2, \mu/2]$. For a polynomial $P \in \mathcal{R}$, $P \leftarrow \beta_\mu$ implies each coefficient of the polynomial is chosen from β_μ independently. For a positive integer x, we define $\mathrm{shiftright}(x, d)$ as $\lfloor x/2^d \rfloor$, the result of d bit shift of x to right. We define $h_1 := \sum_{i=0}^{n-1} 2^{\epsilon_q - \epsilon_p - 1} x^i \in \mathcal{R}_q$, $h_2 := \sum_{i=0}^{n-1}(2^{\epsilon_p - 2} - 2^{\epsilon_p - \epsilon_T - 1} + 2^{\epsilon_q - \epsilon_p - 1})x^i \in \mathcal{R}_q$, and $h := (h_1, \dots, h_1) \in \mathcal{R}_q^k$.

The underlying PKE scheme of Saber is summarized as follows:

- $\mathrm{Gen}(pp)$: $A \leftarrow \mathcal{R}_q^{k \times k}$ and $sk \leftarrow \beta_\mu^k$. Output $pk = (A, B) = (A, \mathrm{shiftright}(A \cdot sk + h, \epsilon_q - \epsilon_p))$ and sk.
- $\mathrm{Enc}(pk, \mu)$: Sample $t \leftarrow \beta_\mu^k$. Output $(U, V) = \big(\mathrm{shiftright}(tA + h, \epsilon_q - \epsilon_p), \mathrm{shiftright}(t \cdot B + h_1 - 2^{\epsilon_p - 1}\mu \bmod p, \epsilon_p - \epsilon_T)\big) \in \mathcal{R}_p^k \times \mathcal{R}_T$.
- $\mathrm{Dec}(sk, (U, V))$: Return $\mu' = \mathrm{shiftright}\big(U \cdot sk - 2^{\epsilon_p - \epsilon_T} \cdot V + h_2 \bmod p, \epsilon_p - 1\big) \in \mathcal{R}_2$.

Review of KR-PCAs against Saber: For LightSaber, we follow the key-recovery attack exploiting a plaintext-checking oracle proposed by Huguenin-Dumittan and Vaudenay [HV20]. For Saber and FireSaber, we follow the key-recovery attack exploiting a plaintext-checking oracle proposed by Osumi, Uemura, Kudo, and Takagi [OUKT21]. Ravi et al. gave a key-recovery side-channel attack against Saber [RRCB20] while they omit the details because Saber is very similar to Kyber. Ngo, Dubrova, Guo, and Johansson [NDGJ21] proposed a key-recovery side-channel attacks against *first-order masked Saber* [BDK+21] and gave a KR-PCA as the building block.

Although their attacks target the Round 2 version, we can mount them against the Round 3 versions since Saber has no changes in Round 3.

KR-PCA against LightSaber: We follow Huguenin-Dumittan and Vaudenay [HV20, Section 6]. We first determine $sk_i = -5, -4, -3, -2, +2, +3, +4, +5$ or some of $-1, 0, +1$ by querying $(U, 2x^i)$ with $U = (u, 0)$ and $u = \{-60/5, -60/4, -60/3, -60/2, +60/2, +60/3, +60/4, +60/5\}$ with guessing plaintext 0. We define $V^+ = \sum_{i \in [0,n): sk_i = 4 \text{ or } 5} 5x^i$ and $V^+ = \sum_{i \in [0,n): sk_i = -4 \text{ or } -5} 5x^i$ and determine whether $sk_i = -1, 0, +1$ by checking $((60, 0), 2x^i + V^+)$ and $((-60, 0), 2x^i + V^-)$ with guessing plaintext 0. Thus, we can determine a coefficient of sk by ten non-adaptive queries to plaintext-checking oracle for LightSaber.

KR-PCA against Saber and FireSaber: Since the principle of the attack is very similar to the KR-PCA against Kyber, we omit the detail of them. Adapting and summarizing the result of Osumi, Uemura, Kudo, and Takagi [OUKT21], we obtain the table of the behavior of decrypted messages in special ciphertexts in Table 11.[22] We can determine a coefficient of sk by eight and six non-adaptive queries to plaintext-checking oracle for Saber and FireSaber, respectively.

[22] They used $U = (u \cdot x^{n-i}, 0^{k-1})$ and $V = t$ to determine sk_i. We adapt it in the form of $U = (u, 0^{k-1})$ and $V = t \cdot x^i$.

Trade-Off: In the case of the skipping-equality-test attack, we can reduce the number of queries as in the case of ntrulpr and Kyber. For example, we can determine first ℓ coefficient by querying (U, V) with $V = \sum_{i=0}^{\ell} 2x^i$ instead of $V = 2x^i$. We can determine ℓ coefficients of sk by ten, eight, and six faulty decapsulation results for LightSaber, Saber, and FireSaber, respectively.

C.5 BIKE

Review of BIKE: BIKE in round 3 [ABB+20] is a KEM scheme based on QC-MDPC [MTSB13], which is a variant of the McEliece PKE upon a code with quasi-cyclic (QC) moderate density parity-check (MDPC) matrix.

Let $\mathcal{R} := \mathbb{F}[x]/(x^r - 1)$. Let $\mathcal{H}_w := \{(h_0, h_1) \in \mathcal{R}^2 \mid \mathsf{HW}(h_0) = \mathsf{HW}(e_1) = w/2\}$. Let $\mathcal{E}_t := \{(e_0, e_1) \in \mathcal{R}^2 \mid \mathsf{HW}(e_0, e_1) = t\}$.

Table 12: Parameter sets of BIKE in Round 3.

parameter sets	r	w	t
BIKE-1	12,323	142	134
BIKE-3	24,659	206	199
BIKE-5	40,973	274	264

The underlying CPA-secure PKE scheme of BIKE is summarized as follows:

- $\mathsf{Gen}(pp)$: $sk := (h_0, h_1) \leftarrow \mathcal{H}_w$. Output $pk = h = h_1 \cdot h_0^{-1} \in \mathcal{R}$ and sk.
- $\mathsf{Enc}(pk, (e_0, e_1) \in \mathcal{E}_t)$: Output $c = e_0 + e_1 h \in \mathcal{R}$.
- $\mathsf{Dec}(sk, c)$: Output $(e_0, e_1) \leftarrow \mathsf{decode}(ch_0, (h_0, h_1))$.

Notice that $ch_0 = e_0 h_0 + e_1 h_1$, which is the syndrome of (e_0, e_1) with the parity-check matrix spanned by h_0 and h_1.

Review of key-recovery attacks against QC-MDPC: Chen, Eisenbarth, von Maurich, and Steinwantdt [CEvMS15] gave DPA-based key-recovery attack against QC-MDPC implemented in a FPGA board. Guo, Johansson, and Stankvoski [GJS16] gave a key-recovery reaction attack against QC-MDPC, which is a variant of the McEliece encryption scheme. They observed that 1) the decryption-failure rates (DFR) for carefuly crafted ciphertexts are strongly related to the distances between 1's in the decryption key and 2) one can reconstruct the decryption key from the knowledge of the distances between 1's. Rossi, Hamburg, Hutter, and Marson [RHHM17] gave DPA-based key-recovery attack against QcBits [Cho16], which is an instantiation of QC-MDPC.

Applicability to BIKE: Huguenin-Dumittan and Vaudenay [HV20] suggested that the GJS attack against QC-MDPC may be applicable to BIKE. However, we do not know whether the GJS attack is applicable to *Round-3* BIKE or not, because Round-3 BIKE is different from QC-MDPC in two points: The one is that BIKE is based on the Niederreiter PKE, while QC-MDPC is based on the McEliece PKE. The other is that BIKE in Round 3 uses the Black-Gray-Flip (BGF) decoder instead of the Black-Gray (BG) decoder. The former difference is not essential, but the latter difference of the decoder would affect the power of the GJS attack.

We here report how the GJS attack is effective for round-3 BIKE.

The GJS attack against QC-MDPC: We first review a simplified version of QC-MDPC.

- $\mathsf{Gen}(pp)$: $sk := (h_0, h_1) \leftarrow \mathcal{H}_w$, which defines a parity-check matrix $H = [H_0|H_1] \in \mathbb{F}_2^{r \times 2r}$ and its systematic form is $[I|\tilde{H}]$ with $\tilde{H} = H_0^{-1} \cdot H_1$. Output $pk = h = h_0^{-1} \cdot h_1 \in \mathcal{R}$ and sk.
- $\mathsf{Enc}(pk, m; (e_0, e_1) \in \mathcal{E}_t)$: Construct a generator matrix $G = [\tilde{H}^\top | I] \in \mathbb{F}_2^{r \times 2r}$ from h. Output $c = mG + (e_0, e_1) \in \mathbb{F}^{2r}$.
- $\mathsf{Dec}(sk, c)$: Output $(e_0, e_1) \leftarrow \mathsf{decode}(c, (h_0, h_1))$. Compute $mG = c - (e_0, e_1)$. Extract m from mG by taking the last r position of mG.

Suppose that τ is even and chosen carefully slightly larger than t to increase the decryption failure. Let

$$\mathcal{P}_{\tau,d} := \left\{ \begin{array}{c} (e_0, 0) \in \mathcal{R}^2 \mid \mathsf{HW}(e_0) = \tau \wedge \mathsf{HW}(e_1) = 0 \wedge \\ \exists \text{ distinct } s_1, \ldots, s_\tau \text{ s.t. } e_0[s_i] = 1 \text{ with } s_{2i} = (s_{2i-1} + d) \bmod r \text{ for } i = 1, \ldots, \tau/2 \end{array} \right\}.$$

The adversary sends a ciphertext crafted by a random sample from $\mathcal{P}_{\tau,d}$ to the decryption oracle and it sees the reaction (yes or no). It can estimate DFR for each $d = 1, \ldots, U$ for the upper bound $U < r/2$. Moreover, they infer $\mu(h_0)$, which is defined as follows: Let $s_1, \ldots, s_\tau$ be the indices such that $h_0[s_i] = 1$. Let $d_r(s_i, s_j)$ be the distance between s_i and s_j, that is, $d_r(s_i, s_j) = \min\{|s_j - s_i|, r - |s_j - s_i|\}$. Define distance profile as

$$\mu(h_0) := \left\{(d, \mu_d) : d \in [0, r), \mu_d = \#\{(s_i, s_j) : 0 \le s_i < s_j < r, d_r(s_i, s_j) = d\}\right\}.$$

Guo et al. gave a key-recovery algorithm from distance profile $\mu(h_0)$ [GJS16, Alg.2]. Paiva and Terada [PT18] also gave a faster key-recovery algorithm from distance profiles $\mu(h_0)$ *and* $\mu(h_1)$.

Applying the GJS attack to BIKE:

Experimental Result: We examine the behavior of DFRs with respect to BGF decoder. We use `BIKE-1`, where $r = 12323$, $w = 142$, and $t = 134$. We pick a key and estimate DFR for random $M = 2000$ plaintexts chosen from $\mathcal{P}_{\tau,d}$ for $U = 1, 2, \ldots, 6162$ with $\tau = 161$ rather than $t = 134$. Figure 17 shows the behavior of DFR for $d = 1, \ldots, 6162$, which varies in the range 0.0–0.5. In figure, we depict the multiplicity 0–4 by blue circle, orange tri-down, green tri-up, red tri-left, and purple tri-right, respectively. For $d = 1, \ldots, 1500$, we can see that the multiplicities are well separated and determine them by estimating DFR for $\mathcal{P}_{\tau,d}$. However, they are mixed in the right-hand side of figure and we cannot determine the multiplicity for large distances. Hence, it seems hard to apply the key-recovery algorithm in the GJS attack (and that in Paiva and Terada [PT18]).

We leave further investigation of key-recovery attack using DFR's behavior as an open problem.

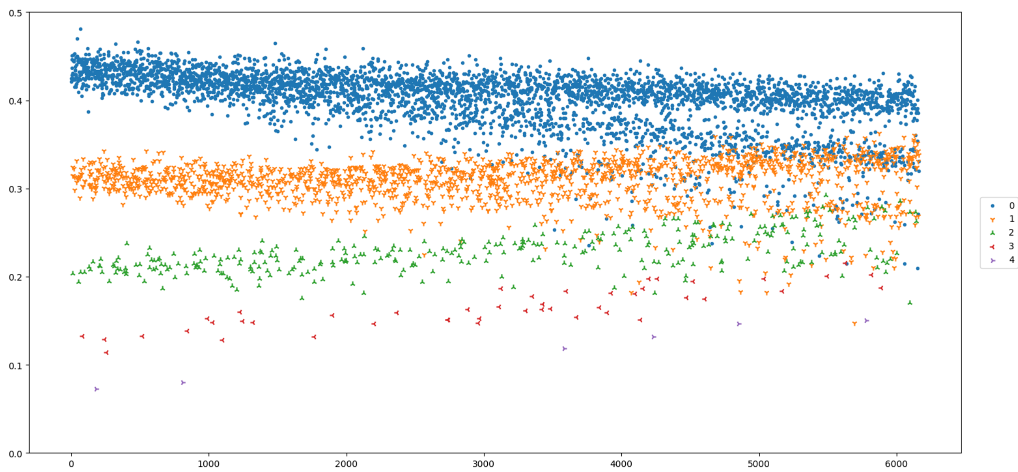

Fig. 17: BIKE: DFR for 2000 plaintexts chosen from $\mathcal{P}_{\tau,d}$ with $\tau = 161$ and $d = 1, 2, \ldots, 6162$.

C.6 FrodoKEM

Review of FrodoKEM: FrodoKEM [NAB+20] is an LWE-based KEM scheme in the alternative candidates.

Let $q = 2^D$ for some $D \le 16$. For a positive integer $B < D$, $\bar{m}$, and $\bar{n}$, they use encode: $\{0,1\}^{B\bar{m}\bar{n}} \to \mathbb{Z}_q^{\bar{m}\times\bar{n}}$ and decode: $\{0,1\}^{B\bar{m}\bar{n}} \to \{0,1\}^{B\bar{m}\bar{n}}$. (Roughly speaking, they compute ec: $k \in [0, 2^B) \mapsto k \cdot q/2^B \in \mathbb{Z}_q$

Table 13: Parameter sets of FrodoKEM in Round 3.

parameter sets	n	q	σ	s	B	$\bar{m}$	$\bar{n}$
Frodo-640	640	2^{15}	2.8	12	2	8	8
Frodo-976	976	2^{16}	2.3	10	3	8	8
Frodo-1344	1344	2^{16}	1.4	6	4	8	8

Table 14: Parameter sets of HQC in Round 3.

parameter sets	r	n_1	k_1	d_1	n_2	k_2	d_2	w	w_e	w_r
hqc-128	17,669	46	16	31	384	8	192	66	75	75
hqc-192	35,851	56	24	32	640	8	320	100	114	114
hqc-256	57,637	90	32	59	640	8	320	131	149	149

and dc: $K \in \mathbb{Z}_q \mapsto \lceil K2^B/q \rfloor \bmod 2^B$ and arrange the result.) Let $\ell = B\bar{m}\bar{n}$ be a message length. They use a distribution χ_s that is a centered symmetric distribution whose support is $\{-s, -(s-1), \dots, s-1, s\}$. (See [NAB+20, Sect.2.2.4 and Table 3] for the concrete distribution.)

The underlying PKE scheme of FrodoKEM [NAB+20] is summarized as follows:

- $\mathsf{Gen}(pp)$: Choose $A \leftarrow \mathbb{Z}_q^{n\times n}$, $S \leftarrow \chi^{n\times\bar{n}}$ and $E \leftarrow \chi^{n\times\bar{n}}$. Output $pk = (A, B) = (A, AS + E)$ and $sk = S$.
- $\mathsf{Enc}(pk, \mu)$: Choose $S', E' \leftarrow \chi^{\bar{m}\times n}$ and $E'' \leftarrow \chi^{\bar{m}\times\bar{n}}$. Output $c = (U, V) = (S'A + E', S'B + E'' + \mathsf{encode}(\mu))$.
- $\mathsf{Dec}(sk = S, (U, V))$: Compute $M = V - U \cdot S$ and output $\mu' = \mathsf{decode}(M)$.

Review of key-recovery attacks against FrodoKEM: There are several KR-PCAs against FrodoKEM by Băetu et al. [BDH+19], Ravi et al. [RRCB20], Vacek and Vàclavek [VV20], and Qin et al. [QCZ+21]. Guo et al. [GJN20] gave a key-recovery timing attack against FrodoKEM.

KR-PCA against FrodoKEM: We only give a rough idea: We have $V - US = \mathsf{encode}(\mu) + \Delta$, where $\Delta = E'' + S'E - E'S \in \mathbb{Z}^{\bar{m}\times\bar{n}}$. The decoding is correct if and only if $-q/2^{B+1} \le \Delta_{i,j} < q/2^{B+1}$ for all i, j [NAB+20, Lamma 2.18]. Thus, we can craft U and V to check S's value directly. For the detail of KR-PCA, see [BDH+19, RRCB20, VV20, QCZ+21].

C.7 HQC

Review of HQC: HQC [AAB+20] is another code-based KEM scheme in the alternative candidates.

Let $\mathcal{R} := \mathbb{F}_2[x]/(x^r - 1)$. Let C be a decodable $[n_1 n_2, k]$ code generated by $G \in \mathbb{F}_2^{k\times n_1 n_2}$, where $n_1 n_2 \le r$. Let decode be a decoder algorithm which corrects an error up to δ. Let $\mathcal{S}_w := \{x \in \mathcal{R} \mid \mathsf{HW}(x) = w\}$. For a polynomial $A = \sum_i a_i x^i \in \mathcal{R}$, we define $\mathsf{trunc}(A, l) = (a_0, \dots, a_{l-1}) \in \mathbb{F}_2^l$.

The underlying PKE scheme of HQC is summarized as follows:

- $\mathsf{Gen}(pp)$: $h \leftarrow \mathcal{R}$. $sk := (x, y) \leftarrow \mathcal{S}_w^2$. Output $pk = (h_0, h_1 = x + h_0 y)$ and sk.
- $\mathsf{Enc}(pk, m \in \mathbb{F}_2^k; (e, f, t) \in \mathcal{S}_{w_e} \times \mathcal{S}_{w_r} \times \mathcal{S}_{w_r})$: Output
$$c = (u, v) = (h_0 t + f, \mathsf{trunc}(h_1 t + e, n_1 n_2) \oplus mG) \in \mathcal{R} \times \mathbb{F}_2^{n_1 n_2}.$$
- $\mathsf{Dec}(sk, (u, v))$: Compute $a = v \oplus \mathsf{trunc}(uy, n_1 n_2) \in \mathbb{F}_2^{n_1 n_2}$ and output $\mathsf{decode}(a)$.

Code in Round 3: In Round 3, the submitter changed the code C from the BCH-Repetition code to RS-RM code. For $m \in \mathbb{F}_2^k \simeq \mathbb{F}_{2^8}^{k_1}$, m is encoded into $m_1 \in \mathbb{F}_{2^8}^{n_1}$ with the Reed-Solomon codes with $[n_1, k_1, d_1]_{2^8}$, then each $m_{1,i} \in \mathbb{F}_{2^8} \simeq \mathbb{F}_2^8$ is encoded into $\tilde{m}_{1,i} \in \mathbb{F}_2^{n_2}$ with the duplicated Reed-Muller code with $[n_2, k_2, d_2]_2$.

Review of key-recovery attacks against HQC: Paiva and Terada [PT19] and Wafo-Tapa, Bettaieb, Bidoux, Gaborit, and Marcatel [WTBB+20] gave key-recovery timing-attacks against HQC. However, their targets are HQC with the code C is the BCH-Repetition code. Huguenin-Dumittan and Vaudenay [HV20] gave a KR-PCA against HQC, which is inspired by a KR-PCA against Lepton [YZ17] by Băetu et al. [BDH+19]. Guo et al. [GJN20] pointed out a potential timing-leakage of the equality test of HQC.

KR-PCA against HQC: We give a rough idea: Let $\tilde{y} = \mathsf{trunc}(y, n_1 n_2)$, the first $n_1 n_2$ coefficients of y. Let us consider $(1, v)$ as a ciphertext and $m_{\text{guess}} = 0^{256}$ and query them to the plaintext-checking oracle. The decryption algorithm will compute $a = v \oplus \mathsf{trunc}(1 \cdot y, n_1 n_2) = v \oplus \tilde{y}$ and decode it into $\mathsf{decode}(a)$. Hence, the plaintext-checking oracle on input $(1, v)$ and 0^{256} tells us if $\mathsf{decode}(v \oplus \tilde{y})$ is decoded into 0^{256} or not.

Băetu et al. [BDH+19, Section 3.5] gave an efficient algorithm to learn $\delta \in \{0,1\}^n$ from the oracle $\mathsf{BOO}(x)$ which returns $\mathsf{boole}(\mathsf{HW}(x \oplus \delta) \leq \rho)$, whose number of queries is at most $n + \lg(n)$. We note that we can use the learning algorithm of Băetu et al. [BDH+19, Section 3.5] for $\{0,1\}^n$, while we consider the Reed-Solomon code over $\mathbb{F}_{2^8}$.

We will mount a two-phase attack as the attacks in [BDH+19, HV20]. Let us consider a string of n_1 packets of n_2 bits. Let $t_1 := (d_1 - 1)/2$ be the maximum Hamming weight of an error that the Reed-Solomon decoder can correct.

1. At first, we learn which packets of $\tilde{y}$ contain an error using the learning algorithm of Băetu et al.with $n_1 + \lg(n_1)$ queries.
   Each packet represents 0 and 1 by 0^{n_2} and 1^{n_2}, which can be considered as the codeword of the duplicated Reed-Muller code corresponding to 0 and 1 in $\mathbb{F}_2^8$, respectively. Notice that $\tilde{y}$'s Hamming weight is at most w, since y is in $\mathcal{S}_w$, and d_2 are larger than the double of w. Thus, each packet of $\tilde{y}$ is decoded into 0 originally and the packet xored by 1^{n_2} is decoded into 1. The Reed-Solomon decoder will fail to decode the received word into 0 if the received word contains at least $t_1 + 1$ 1's. Thus, we can use the learning algorithm for $\{0,1\}^{n_1}$.
2. Next, for each packet, we modify $\tilde{y}$ to have exactly $t_1 - 1$ incorrect other packets and apply the learning algorithm on the packet with the threshold of the duplicate Reed-Muller codes. This requires $n_2 + \lg(n_2)$ queries on each packet. After that we know $\tilde{y}$.
3. Finally, we compute the last rest $n - n_1 n_2$ bits of y by checking if $h_1 - h_0 y \in \mathcal{S}_w$ or not. This is done by brute force on $2^{n - n_1 n_2}$ candidates.

C.8 NTRU Prime

ntrulpr of NTRU Prime: Since we already gave the key-recovery attack using the plaintext-checking oracle in section 4, we omit the details.

sntrupr of NTRU Prime:

sntrupr of NTRU Prime: Streamlined NTRU Prime (sntrupr) has parameter sets p, q, and w. p and q are prime numbers and w is a positive integer. We note that $2p \geq 3w$ and $q \geq 16w + 1$. They choose $q = 6q' + 1$ for some q'. For concrete values, see Table 15.

Table 15: Parameter sets of sntrupr of NTRU Prime

parameter sets	p	q	w
`sntrupr653`	653	4621	288
`sntrupr761`	761	4591	286
`sntrupr857`	857	5167	322
`sntrupr953`	953	6343	396
`sntrupr1013`	1013	7177	448
`sntrupr1277`	1277	7879	492

Let $\mathcal{R} := \mathbb{Z}[x]/(x^p - x - 1)$ and $\mathcal{R}_a := (\mathbb{Z}/a)[x]/(x^p - x - 1)$ for $a = 3, q$. Let $\mathcal{S} := \{a = \sum_{i=0}^{p-1} a_i x^i \in \mathcal{R} \mid a_i \in \{-1, 0, +1\}, \mathsf{HW}(a) = w\}$, a set of "short" polynomials. For $a \in [-(q-1)/2, (q-1)/2]$, define $\mathsf{Round}(a) = 3 \cdot \lceil a/3 \rfloor$.[23]

The underlying CPA-secure PKE scheme[24] works as follows:

[23] When $q = 6q' + 1$, $\mathsf{Round}([-(q-1)/2, (q-1)/2]) \in [-(q-1)/2, (q-1)/2]$.
[24] 'Streamlined NTRU Prime Core' in the specification.

– $\mathsf{Gen}(pp)$: Choose $g \leftarrow \mathcal{R}$ that satisfies $g \in \mathcal{R}_3^\times$ at random. Compute $1/g \in \mathcal{R}_3$. Choose $f \leftarrow \mathcal{S}$. Compute $h = g/(3f) \in \mathcal{R}_q$. Output $pk = h$ and $sk = (f, 1/g)$.
– $\mathsf{Enc}(pk, r \in \mathcal{S})$: Compute $hr \in \mathcal{R}_q$ and output $c = \mathsf{Round}(hr \bmod^\pm q)$.
– $\mathsf{Dec}(sk = (f, v), c)$: Compute $e = (3fc \bmod^\pm q) \bmod^\pm 3$. Compute $r' = ev =\!\bmod^\pm 3$. Output r' if $\mathsf{HW}(r') = w$. Otherwise, output $r' = (1, 1, \dots, 1, 0, \dots, 0)$ with $\mathsf{HW}(r') = w$.

Due to rounding, we have a 'short' error m such that $c = hr + m$.

KR-PCA against NTRU Prime: One might consider that designing a key-recovery plaintext-checking attack for sntrupr of NTRU Prime is easy by following the key-recovery plaintext-checking attacks for NTRU-HPS [DDS+19] and that for NTRU-HRSS [ZCQD21].

However, there are several obstacles and we fail to design the KR-PCA.

The first obstacle is that it is hard to fix $m = 0$, since m is determined by hr and $\mathsf{Round}(hr)$.

Even if we could fix $m = 0$, there are more obstacles. Notice that Dec checks the Hamming weight of r'. In the KR-PCAs against NTRU, they use small-weight plaintext $(r, 0)$ to check the information of the secret key. The Hamming-weight check judges such plaintexts invalid.

Very recently, Ravi, Ezerman, Bhasin, Chattopadhyay, and Roy [REB+21] proposed a key-recovery plaintext-checking attack against sntrupr of NTRU Prime by extending the key-recovery attack by Jaulmes and Joux [JJ00], which they call a key-recovery attack using the decryption failure oracle. They mount their attack against `sntrupr761`.

C.9 SIKE

Brief Review of SIKE: SIKE [JAC+20] is KEM scheme based on SIDH [JD11, DJP14]. For a survey of isogeny-based cryptography, we recommend reading [Cos21].

Let $p = 2^{e_2}3^{e_3} - 1$. Let E be a supersingular elliptic curve over $\mathbb{F}_{p^2}$. Let $P_2, Q_2 \in E[2^{e_2}]$ and $P_3, Q_3 \in E[3^{e_3}]$ linearly independent points of order 2^{e_2} and 3^{e_3} respectively. Let $\{0,1\}^n$ be a message space and let $\mathsf{F} : \mathbb{F}_{p^2} \to \{0,1\}^n$ be a random oracle.

Roughly speaking, the underlying PKE scheme [JAC+20, Algorithm 1] is summarized as follows (for the details, see the specification):

– $\mathsf{isogen}_\ell(sk_\ell)$ with $(m, \ell) = (2, 3)$ or $(3, 2)$: On input $sk_\ell \in [0, \ell^{e_\ell})$, compute $S = P_\ell + [sk_\ell]Q_\ell$, compute isogeny $\phi_\ell : E \to E/\langle S \rangle$, and compute $E'_m := E/\langle S \rangle = \phi_\ell(E)$. Compute $P'_m = \phi_\ell(P_m)$ and $Q'_m = \phi_\ell(Q_m)$. Output (E'_m, P'_m, Q'_m).[25]
– $\mathsf{isoex}_\ell(pk_m, sk_\ell)$ with $(m, \ell) = (2, 3)$ or $(3, 2)$: On input $pk_m = (E'_\ell, P'_\ell, Q'_\ell)$ and $sk_\ell \in [0, \ell^{e_\ell})$, compute $S = P'_\ell + [sk_\ell]Q'_\ell$ and compute $E'' = E'_m/\langle S \rangle = E/\langle \phi_m(P_\ell + [sk_\ell]Q_\ell) \rangle$. Compute j_ℓ as the j-invariant of E''.
– $\mathsf{Gen}(pp)$: Choose $sk_3 \leftarrow [0, 3^{e_3})$ and $pk_3 \leftarrow \mathsf{isogen}_3(sk_3)$. Output pk_3 and sk_3.
– $\mathsf{Enc}(pk_3, \mu)$: Choose $sk_2 \leftarrow [0, 2^{e_2})$ and $c_2 \leftarrow \mathsf{isogen}_2(sk_2)$. Compute $j = \mathsf{isoex}_2(pk_3, sk_2)$. Compute $c' = \mathsf{F}(j) \oplus \mu$. Output (c_2, c').
– $\mathsf{Dec}(sk_3, (c_2, c'))$: Compute $j' = \mathsf{isoex}_3(c_2, sk_3)$ and output $\mu' = c' \oplus \mathsf{F}(j')$.

Review of key-recovery attacks against SIDH/SIKE: Galbraith, Petit, Shani, and Ti [GPST16] gave a key-recovery key-mismatch oracle against SIDH with a fixed key. Koziel, Azarderakhsh, and Jao [KAJ17] gave a key-recovery SCA against SIDH with a fixed key and a PKE version of SIDH, which is equivalent to SIKE). Gélin and Wesolowski [GW17] gave a key-recovery FIA against SIDH with fixed key and the PKE version of SIDH, which uses *loop-abort* fault-injection to stop iterating computation of elliptic curves. Ti [Ti17] gave a key-recovery FIA against SIDH with fixed secret and signature schemes based on SIDH, which queries a random point X, obtains $\phi(X)$ for a secret isogeny ϕ, and recovers ϕ. Thus, it cannot be used in the context of a key-recovery attack against PKE.

[25] Correctly speaking, this algorithm outputs $(P'_m, Q'_m, R'_m = P'_m - Q'_m)$ and omits E'_m. We can reconstruct E'_m from P'_m, Q'_m, and R'_m.

$$\begin{tikzcd}
 & \begin{array}{l} \underline{pk_3 = (E_2', P_2', Q_2'):} \\ E_2' = E/\langle P_3 + [sk_3]Q_3 \rangle = \phi_3(E) \\ P_2' = \phi_3(P_2),\, Q_2' = \phi_3(Q_2) \end{array} \arrow[r, Rightarrow, "sk_2"] & \begin{array}{l} E/\langle \phi_3(P_2 + [sk_2]Q_2) \rangle \\ \quad = E/\langle P_2' + [sk_2]Q_2' \rangle \end{array} \arrow[dr, Rightarrow] & \\
pp = (E, P_2, Q_2, P_3, Q_3) \arrow[ur, dashed, "sk_3"] \arrow[dr, Rightarrow, "sk_2"'] & & & j = j(E_{\mathsf{shared}}) \\
 & \begin{array}{l} \underline{c_2 = (E_3', P_3', Q_3'):} \\ E_3' = E/\langle P_2 + [sk_2]Q_2 \rangle = \phi_2(E) \\ P_3' = \phi_2(P_3),\, Q_3' = \phi_2(Q_3) \end{array} \arrow[r, dashed, "sk_3"] & \begin{array}{l} E/\langle \phi_2(P_3 + [sk_3]Q_3) \rangle \\ \quad = E/\langle P_3' + [sk_3]Q_3' \rangle \end{array} \arrow[ur, dashed] &
\end{tikzcd}$$

Fig. 18: Diagram of the underlying KE scheme of SIKE.

KR-PCA against the underlying PKE scheme of SIKE: We follow the GPST attack proposed by Galbraith, Petit, Shani, and Ti [GPST16] and adapt it to the PKE scheme. They explicitly write down a key-recovery attack using the key-mismatch oracle against SIDH, where its secret key is sk_2. They confirmed their attack is easily applicable to the case of sk_3 in [GPST16, Remark 2]. For the case of sk_3, see e.g., a writeup by goulov and mandlebro on the problem sidhe in PlaindCTF 2020 [gm20]. We here review how the attack works, because there are some differences. For example, we need the plaintext-checking oracle instead of the key-mismatch oracle; we do not need the scale-up factor θ in the attacks, since the uncompressed SIKE does not involve the pairing checks.

In order to count the number of queries, we briefly review how the attack extracts sk_3. Suppose that we know the first i trits of sk_3 and write sk_3 as

$$sk_3 = k_i + s_i 3^i + s' 3^{i+1},$$

where $k_i \in [0, 3^{i-1})$ is known, $s_i \in \{0, 1, 2\}$, and $s' \in \mathbb{Z}$ are unknown.

We generate $c_2 = (E_3', P_3', Q_3')$, where $P_3' = \phi_2(P_3)$ and $Q_3' = \phi_2(Q_3)$, $E/\langle P_2' + [sk_2]Q_2' \rangle$, and j-invariant $j_k = j(E/\langle P_2' + [sk_2]Q_2' \rangle)$, and $k = \mathsf{F}(j_k)$ as the encryption. In order to recover $s_i \in \{0, 1, 2\}$, we compute

$$c^{(z)} = (E_3', P^{(z)}, Q^{(z)}) = \left(E_3', P_3' - [3^{e_3 - i - 1}(k_i + z3^i)]Q_3', [1 + 3^{e_3 - i - 1}]Q_3'\right)$$

for $z = 0, 1, 2$ and query $(c^{(z)}, k)$ with guessing plaintext 0^n.

The decryption algorithm first computes

$$\begin{aligned}
S^{(z)} &= P^{(z)} + [sk_3]Q^{(z)} \\
&= P_3' - [3^{e_3-i-1}(k_i + z3^i)]Q_3' + [sk_3][1 + 3^{e_3-i-1}]Q_3' \\
&= P_3' + [sk_3]Q_3' + [-3^{e_3-i-1}(k_i + z3^i) + 3^{e_3-i-1}(k_i + s_i 3^i + s' 3^{i+1})]Q_3' \\
&= P_3' + [sk_3]Q_3' + [(s_i - z)3^{e_3-1}]Q_3',
\end{aligned}$$

where we used the fact that P_3' and Q_3' are of order 3^{e_3}. The subgroups $\langle P_3' + [sk_3]Q_3' \rangle$, $\langle P_3' + [sk_3]Q_3' + [3^{e_3-1}]Q_3' \rangle$, and $\langle P_3' + [sk_3]Q_3' + [2 \cdot 3^{e_3-1}]Q_3' \rangle$ are distinct and, (heuristically speaking), $j_k = j(E/\langle P_3' + [sk_3]Q_3' + [(s_i - z)3^{e_3-1}]Q_3' \rangle)$ if and only if $s_i = z$. If $s_i = z$, then the decryption algorithm obtains the plaintext 0^n and the PCO returns 1. Otherwise, it will obtain the plaintext $\mathsf{F}(j_k) \oplus \mathsf{F}(j(E/\langle P_3' + [sk_3]Q_3' + [(s_i - z)3^{e_3-1}]Q_3' \rangle))$, which is not 0^n heuristically, and the PCO returns 0.

In the context of the skipping-equality-test attack, we will check $K = \mathsf{KDF}(0^k, ct)$ or not. Summarizing the above, we can determine $s_i \in \{0, 1, 2\}$ for $i = 0, \dots, e_3 - 1$ by sending two queries with $z = 0$ and 1 and guessing key K and checking the returned value.

Table of Contents