Kleptographic Attacks against Implicit Rejection

Antoine Joux [1]   Julian Loss [1]   Benedikt Wagner [1,2]

February 16, 2024

[1] CISPA Helmholtz Center for Information Security
{joux,loss,benedikt.wagner}@cispa.de
[2] Saarland University

Abstract

Given its integral role in modern encryption systems such as CRYSTALS-Kyber, the Fujisaki-Okamoto (FO) transform will soon be at the center of our secure communications infrastructure. An enduring debate surrounding the FO transform is whether to use explicit or implicit rejection when decapsulation fails. Presently, implicit rejection, as implemented in CRYSTALS-Kyber, is supported by a strong set of arguments. Therefore, understanding its security implications in different attacker models is essential.

In this work, we study implicit rejection through a novel lens, namely, from the perspective of kleptography. Concretely, we consider an attacker model in which the attacker can subvert the user's code to compromise security while remaining undetectable. In this scenario, we present *three attacks* that significantly reduce the security level of the FO transform with implicit rejection. Notably, our attacks apply to CRYSTALS-Kyber.

Keywords: Kleptography, Implicit Rejection, Chosen-Ciphertext Security, Fujisaki-Okamoto Transform, Kyber

Contents

1 Introduction **3**
1.1 Our Contribution . . . 4
1.2 More on Related Work . . . 4
1.3 Outline . . . 5

2 Preliminaries **5**

3 Kleptographic Model **7**

4 Our Attacks **10**
4.1 Subverting Decapsulation Only . . . 10
4.2 Subverting Key Generation Only . . . 13
4.3 An Attack with Preprocessing . . . 15

A Script for Attack Complexity and Advantage **24**

1 Introduction

In response to the threat posed by large-scale quantum computers, NIST hosted a competition to standardize quantum-secure cryptography [NIS17], including key encapsulation mechanisms (KEMs) and digital signatures. The design underlying a large portion of the submitted KEMs follows a two-step approach:

1. One constructs a public key encryption scheme (PKE) secure against chosen-plaintext attacks (CPA) [GM84]. Concretely, one-wayness (OW-CPA) is required.

2. CPA security is lifted to obtain a KEM secure against chosen-ciphertext attacks (IND-CCA) [NY90, RS92]. This is accomplished using a generic transformation in the random oracle model due to Fujisaki and Okamoto [FO99].

For example, this approach is taken by CRYSTALS-Kyber [SAB+22, ABD+21], a winner of the competition that has been selected for standardization.

Fujisaki-Okamoto. The Fujisaki-Okamoto (FO) transform [FO99, HHK17] generically transforms OW-CPA security into IND-CCA security, in the (quantum) random oracle model. Specifically, assuming a OW-CPA secure PKE, the FO transform yields an IND-CCA secure KEM. In simplified terms, this KEM is constructed as follows: To encapsulate a key, a random string m is chosen and encrypted via PKE. Crucially, the encryption coins are deterministically derived from m. The encapsulated key is then pseudorandomly derived from m and the ciphertext. During decapsulation, the ciphertext is first decrypted to obtain m, followed by a consistency check through re-encryption. Decapsulation only derives and outputs K if this consistency check succeeds.

Building on this basic template, several variants of the FO transform have been proposed and analyzed, e.g., [Den03, HHK17]. Given that the FO transform will soon be the backbone of post-quantum secure communication, it is essential to have a clear understanding of these variants in different attacker models.

Implicit vs. Explicit Rejection. A key point of discussion in the context of the FO transform is the behavior of decapsulation when the consistency check fails. Specifically, two approaches are debated: The FO transformed KEM with *explicit rejection* outputs a dedicated failure symbol $\bot$. Contrary to that, the *implicit rejection* variant outputs an implicit rejection key K which is pseudorandomly derived from the ciphertext and a secret seed. This seed is only used for implicit rejection and can be thought of as a secondary secret key. While explicit rejection seems more intuitive, implicit rejection is widely preferred, which is evidenced, for instance, by its adoption in Kyber. This preference mostly stems from tighter security bounds for implicit rejection in the quantum random oracle model (QROM) [BDF+11]: a long line of work [SXY18, JZC+18, BHH+19, HKSU20, KSS+20] has improved the tightness for the implicit rejection variant, while progress for explicit rejection has been more recent [DFMS22, HHM22].

The goal of this work is to deepen our understanding of the security implications associated with implicit and explicit rejection. To this end, we extend the study of rejection to the context of subversion. Specifically, we want to understand if implicit rejection gives additional leverage to a powerful attacker who can manipulate a user's code to attack users.

Kleptography. Kleptography dates back to Young and Yung in the 1990s [YY96, YY97a] and considers the strong attacker model outlined above. In essence, a kleptographic attacker can manipulate or fully replace a user's code of a cryptographic system. The primary goal of such an attacker is twofold: firstly, the attack should be *successful*. That is, the attacker successfully breaches the security of any victim utilizing the manipulated cryptosystem, e.g., gaining access to encrypted messages. Secondly, the attack must remain *undetectable*. Concretely, the victim should not be able to tell apart the manipulated code from the benign code when having black-box access. To accomplish this task, the kleptographic attacker itself utilizes cryptography.

Achieving these objectives renders kleptographic attacks challenging yet exceedingly valuable for state-level actors or malevolent vendors. Not even expert users carefully check the code they use, and even if they were inclined to do so, cryptographic code is often obfuscated or stored in secure modules. Consequently, as long as a kleptographic attack is undetectable as outlined above, users have no chance to discern malicious code.

One can further strengthen the undetectability requirement by demanding that no other actor B apart from the kleptographic attacker A can carry out the attack on A's behalf, even if B can detect

Attack	Subvert	Public Key	Memory	Time Offline	Time Online	Advantage
Section 4.1	Decaps	✓	2^8	2^0	2^2	0.997
Section 4.2	Key Gen	✗	2^7	2^0	2^{130}	0.999
Section 4.3	Key Gen	✗	2^{111}	2^{154}	2^{106}	0.692

Table 1: Overview of the kleptographic attacks that we introduce against implicit rejection, applied to Kyber [ABD+21]. We assume that Kyber has spreadness $\gamma \leq 1/1000$. The complexities and advantage are computed using a Python script given in Appendix A.

the presence of A in the victim's system. To motivate this stronger undetectability notion, consider a scenario where the attack is deployed widely, targeting a large user base. The kleptographic attacker, say an intelligence agency A, wants to use this to its advantage over a second agency B. Now, envision agency B is investing substantial resources to reverse-engineer the compromised code and extract all embedded information. Ideally, agency A would hope that even with this information, agency B cannot carry out the attack. We refer to this strong notion of undetectability as *public key*.

1.1 Our Contribution

In this work, we study implicit rejection in the Fujisaki-Okamoto transform from the perspective of kleptography. Concretely, we develop three ways to subvert parts of the resulting KEM code that significantly reduce the security level of the KEM while being undetectable. All of our attacks only tamper with code related to the implicit rejection path:

- *Subverting Decapsulation.* In our first attack, we subvert the implicit rejection path of the decapsulation algorithm by making the implicit rejection key depend on the victim's secret key. This attack is public key: even given all information embedded by the attacker in the subverted code, one can not distinguish the subverted algorithm from the honest one. Especially, a user who reverse-engineers the subverted algorithm will not have a significant advantage in breaking the scheme for other subverted users.
- *Subverting Key Generation.* Our second attack does not modify the code of decapsulation. Instead, we show how to modify the code generating the seed s to leak information about the actual secret key. This attack embeds the attacker's secret into the subverted algorithm and is undetectable as long as the code is not reverse-engineered.
- *Preprocessing.* As a variant of our second attack, we show how to leverage a variant of Hellman tables [Hel80] to speed up the online phase of the attack.

To exemplify the complexity and success probabilities of our attacks, we consider Kyber [ABD+21] as a running example. We present the results in Table 1. Notably, our first attack results in a complete break of Kyber, whereas our second and third attack halve the desired security level of 256 bit against classical adversaries. A quantum adversary can further speed up the running time of our second attack using two applications of Grover's algorithm [Gro96], in which case about 64 bits of quantum security would remain.

The takeaway from our results is a vulnerability of implicit rejection: it can serve as a side channel that is remarkably challenging to identify. Contrary to popular belief, implicit rejection is not necessarily superior in every aspect.

1.2 More on Related Work

Here, we discuss related work, including works on the Fujisaki-Okamoto transform, implicit vs. explicit rejection, and kleptography.

Fujisaki-Okamoto Transform. The Fujisaki-Okamoto (FO) transform has been introduced originally by Fujisaki and Okamoto [FO99, FO13] to generically construct chosen-ciphertext secure public key encryption in the random oracle model. The version that results in a chosen-ciphertext secure key encapsulation mechanism and is now the standard is due to Dent [Den03]. In his PhD thesis [Per12], Persichetti is the first to propose implicit rejection for the FO transform. Hofheinz, Hövelmanns, and Kiltz [HHK17] gave a modular analysis of the FO transform in presence of correctness errors. They analyze both implicit and explicit rejection in the random oracle model. Additionally, they propose a variant of the FO transform that they analyze in the quantum random oracle model (QROM) [BDF+11]. Later, the FO transform with implicit rejection has also been analyzed in the QROM [JZC+18] with a non-tight security bound. By including public keys as input for the hash fucntions, Duman et al. [DHK+21] have improved the security bounds for multi-user security both in the ROM and QROM. Their techniques work for both explicit and implicit rejection.

Implicit vs. Explicit Rejection. In the work of Hofheinz, Hövelmanns, and Kiltz [HHK17], implicit rejection (as opposed to explicit rejection) does not require the underlying to have *spreadness*, which can result in more efficient parameters. Since then, a long line of work has analyzed the FO transform and its variants in the QROM [SXY18, JZC+18, BHH+19, HKSU20, KSS+20]. These works improve the concrete security bound significantly, but most of the techniques only apply to implicit rejection. The recent work of Hövelmanns, Hülsing, and Majenz [HHM22] improves the QROM security bound for the explicit rejection variant of the FO transform, thereby decreasing the tightness gap. Implicit rejection is implemented in Kyber, where it is argued that this has the practical advantage that "implementations of Kyber's decapsulation are safe to use even if higher level protocols fail to check the return value" [ABD+21].

Kleptography. Kleptography has been introduced and first studied by Young and Yung [YY96, YY97a]. Early works [YY96, YY97a, YY97b, YY01] present attacks against RSA, DSA, the Diffie-Hellman key exchange, and more. All of these works use rather informal definitions to model kleptographic attacks. With the Snowden revelations, there has been a renewed focus on kleptography under the term *algorithm-substitution attacks* [BPR14, DFP15, BJK15]. These works include attacks on symmetric encryption schemes and a formal model for attacks and what it means to resist a subversion attack. The same has been done for signature schemes [AMV15]. Cryptographic constructions secure against kleptography (what is called the *complete subversion model*) have further been studied in [RTYZ16, TY17, RTYZ17]. Examples of partial subversion of cryptographic systems include randomness subversion [BBN+09, BH15] or subversion of common reference strings [BFS16, Fuc18].

Kleptography against Post-Quantum KEMs. Ravi et al. present kleptographic attacks against Kyber [RBC+22]. Their work uses properties specific to lattices to introduce a backdoor in the key generation algorithm. In contrast to that, our work treats the FO transform generically and we only use Kyber as a running example.

1.3 Outline

We structure the rest of the paper as follows. First, we recall the Fujisaki-Okamoto transform [FO99, Den03, HHK17] and other important preliminaries in Section 2. Then, in Section 3, we introduce the formal model in which we will present and analyze our kleptographic attacks. In Section 4, we present and analyze our attacks.

2 Preliminaries

Here, we fix common notation and recall the Fujisaki-Okamoto transform [FO99, HHK17].

Notation. For a finite set S, we write $s \xleftarrow{\$} S$ when s is sampled uniformly at random from S. For a probabilistic algorithm Alg, we write $y := \mathsf{Alg}(x;\rho)$ to denote the process of running Alg with random coins ρ on input x and storing the output in y. If ρ is sampled uniformly at random, We write $y \leftarrow \mathsf{Alg}(x)$ when the random coins ρ are sampled uniformly at random. The notation $y \in \mathsf{Alg}(x)$ means that there are coins ρ such that $\mathsf{Alg}(x;\rho)$ outputs y. We denote the running time of Alg by $\mathbf{T}(\mathsf{Alg})$.

Cryptographic Primitives. We recall relevant cryptographic primitives, namely, public key encryption (PKE) and key encapsulation mechanisms (KEMs). We start with public key encryption.

Definition 1 (Public Key Encryption)**.** A public key encryption scheme with public key space $\{0,1\}^{\ell_p}$, secret key space $\{0,1\}^{\ell_s}$, message space $\{0,1\}^{\ell_m}$, randomness space $\{0,1\}^{\ell_r}$, and ciphertext space $\{0,1\}^{\ell_c}$ is a triple $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ of algorithms with the following syntax:

- $\mathsf{Gen} \to (\mathsf{pk}, \mathsf{sk})$ does not take any input and outputs a public key $\mathsf{pk} \in \{0,1\}^{\ell_p}$ and a secret key $\mathsf{sk} \in \{0,1\}^{\ell_s}$.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{m}) \to c$ takes as input a public key $\mathsf{pk} \in \{0,1\}^{\ell_p}$ and a message $\mathsf{m} \in \{0,1\}^{\ell_m}$, and outputs a ciphertext $c \in \{0,1\}^{\ell_c}$. We assume it uses randomness $\rho \in \{0,1\}^{\ell_r}$ and write $\mathsf{Enc}(\mathsf{pk}, \mathsf{m}; \rho)$ to make the randomness explicit.
- $\mathsf{Dec}(\mathsf{sk}, c) \to \mathsf{m}$ is deterministic, takes as input a secret key $\mathsf{sk} \in \{0,1\}^{\ell_s}$ and a ciphertext $c \in \{0,1\}^{\ell_c}$, and outputs a message $\mathsf{m} \in \{0,1\}^{\ell_m}$ or $\perp$.

Further, we say that the scheme has correctness error $\delta \in [0,1]$, if we have

$$\mathop{\mathbb{E}}_{(\mathsf{pk},\mathsf{sk})}\left[\max_{\mathsf{m}} \Pr_{c}\left[\mathsf{Dec}(\mathsf{sk}, c) \neq \mathsf{m}\right]\right] \leq \delta,$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$, $\mathsf{m} \in \{0,1\}^{\ell_m}$, and $c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m})$.

The next definition, following [FO99, HHK17], introduces spreadness of a public key encryption scheme. Intuitively, if spreadness is large, then ciphertexts have high min-entropy. We can naturally assume that schemes have large spreadness, i.e., it is unlikely that a fixed ciphertext is hit when encrypting. In the context of our attacks, we will sample random ciphertexts and submit them to the decapsulation oracle. We will need to bound the probability that such a ciphertext is valid, i.e., does not trigger implicit rejection. Spreadness will be useful for this.

Definition 2 (Spreadness)**.** Let $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme with randomness space $\{0,1\}^{\ell_r}$, message space $\{0,1\}^{\ell_m}$, and ciphertext space $\{0,1\}^{\ell_c}$. We say that PKE has spreadness $\gamma \in [0,1]$, if

$$\mathop{\mathbb{E}}_{(\mathsf{pk},\mathsf{sk})}\left[\max_{\mathsf{m},c} \Pr_{\rho}\left[\mathsf{Enc}(\mathsf{pk}, \mathsf{m}; \rho) = c\right]\right] \leq \gamma,$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$, $\mathsf{m} \in \{0,1\}^{\ell_m}, c \in \{0,1\}^{\ell_c}$, and $\rho \xleftarrow{\$} \{0,1\}^{\ell_r}$.

As a tool in our first attack, we need a public key encryption for which ciphertexts are indistinguishable from random strings. There are many natural examples of such schemes, e.g., [ElG84, Reg05].

Definition 3 (IND-CPA-R Security)**.** Let $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme with message space $\{0,1\}^{\ell_m}$ and ciphertext space $\{0,1\}^{\ell_c}$. Let $\mathcal{D}$ be an algorithm. The IND-CPA-R advantage of PKE is defined to be

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA\text{-}R}}_{\mathcal{D},\mathsf{PKE}} := \left|\Pr\left[\mathbf{IND\text{-}CPA\text{-}R}^{\mathcal{D}}_{\mathsf{PKE},0} \Rightarrow 1\right] - \Pr\left[\mathbf{IND\text{-}CPA\text{-}R}^{\mathcal{D}}_{\mathsf{PKE},1} \Rightarrow 1\right]\right|,$$

where game **IND-CPA-R** is given in Figure 1.

Game $\mathbf{IND\text{-}CPA\text{-}R}^{\mathcal{D}}_{\mathsf{PKE},0}$	**Game** $\mathbf{IND\text{-}CPA\text{-}R}^{\mathcal{D}}_{\mathsf{PKE},1}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$	$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$
$(\mathsf{m}, St) \leftarrow \mathcal{D}(\mathsf{pk})$	$(\mathsf{m}, St) \leftarrow \mathcal{D}(\mathsf{pk})$
if $\mathsf{m} \notin \{0,1\}^{\ell_m}$: **return** 0	**if** $\mathsf{m} \notin \{0,1\}^{\ell_m}$: **return** 0
$c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m})$	$c \xleftarrow{\$} \{0,1\}^{\ell_c}$
return $b \leftarrow \mathcal{D}(St, c)$	**return** $b \leftarrow \mathcal{D}(St, c)$

Figure 1: The IND-CPA-R game **IND-CPA-R** for a distinguisher $\mathcal{D}$ and a public key encryption scheme $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\{0,1\}^{\ell_m}$ and ciphertext space $\{0,1\}^{\ell_c}$.

Next, we turn to the definition of key encapsulation mechanisms, which are closely related to public key encryption. Intuitively, we can think of a key encapsulation mechanism as encrypting a random key. Conversely, we can obtain public key encryption by combining a key encapsulation mechanism with symmetric encryption.

Definition 4 (Key Encapsulation Mechanism)**.** A key encapsulation mechanism with public key space $\{0,1\}^{\ell_p}$, secret key space $\{0,1\}^{\ell_s}$, randomness space $\{0,1\}^{\ell_r}$, ciphertext space $\{0,1\}^{\ell_c}$, and key space $\{0,1\}^{\ell_k}$ is a triple $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ of algorithms with the following syntax:

- $\mathsf{Gen} \to (\mathsf{pk}, \mathsf{sk})$ does not take any input and outputs a public key $\mathsf{pk} \in \{0,1\}^{\ell_p}$ and a secret key $\mathsf{sk} \in \{0,1\}^{\ell_s}$.
- $\mathsf{Encaps}(\mathsf{pk}) \to (K, c)$ takes as input a public key $\mathsf{pk} \in \{0,1\}^{\ell_p}$ and outputs a key $K \in \{0,1\}^{\ell_k}$ and a ciphertext $c \in \{0,1\}^{\ell_c}$. We assume it uses randomness $\rho \in \{0,1\}^{\ell_r}$ and write $\mathsf{Encaps}(\mathsf{pk}; \rho)$ to make the randomness explicit.
- $\mathsf{Decaps}(\mathsf{sk}, c) \to K$ is deterministic, takes as input a secret key $\mathsf{sk} \in \{0,1\}^{\ell_s}$ and a ciphertext $c \in \{0,1\}^{\ell_c}$, and outputs a key $K \in \{0,1\}^{\ell_k}$ or $\perp$.

Further, we say that the scheme has correctness error $\delta \in [0,1]$, if we have

$$\Pr\left[\mathsf{Decaps}(\mathsf{sk}, c) \neq K \mid (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen},\ (K, c) \leftarrow \mathsf{Encaps}(\mathsf{pk})\right] \leq \delta.$$

Fujisaki-Okamoto Transform. The Fujisaki-Okamoto transform [FO99, Den03, HHK17] turns a public key encryption scheme into a key encapsulation mechanism. It is used to generically achieve indistinguishability under chosen-ciphertext attacks. More precisely, let $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme with public key space $\{0,1\}^{\ell_p}$, secret key space $\{0,1\}^{\ell_s}$, message space $\{0,1\}^{\ell_m}$, randomness space $\{0,1\}^{\ell_r}$, and ciphertext space $\{0,1\}^{\ell_c}$. Let $\lambda, \ell_k \in \mathbb{N}$ be parameters and $\mathsf{G}\colon \{0,1\}^* \to \{0,1\}^{\ell_r}$ and $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^{\ell_k}$ be random oracles. Then, the result of the Fujisaki-Okamoto transform is a key encapsulation mechanism $\mathsf{KEM}[\mathsf{PKE}, \mathsf{H}, \mathsf{G}]$ with public key space $\{0,1\}^{\ell_p}$, secret key space $\{0,1\}^{\ell_s+\lambda}$, randomness space $\{0,1\}^{\ell_m}$, ciphertext space $\{0,1\}^{\ell_c}$, and key space $\{0,1\}^{\ell_k}$ We present $\mathsf{KEM}[\mathsf{PKE}, \mathsf{H}, \mathsf{G}]$ in Figure 2. Importantly, we focus on the variant with implicit rejection following [HHK17]. That is, instead of returning $\perp$ when an invalid ciphertext is input into the decapsulation algorithm, the algorithm returns a pseudorandom key $\mathsf{H}(s, c)$ based on the ciphertext and a secret seed $s \in \{0,1\}^\lambda$.

Alg Gen
- $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{PKE.Gen}$, $\quad s \xleftarrow{\$} \{0,1\}^\lambda$
- $\mathsf{sk} := (\mathsf{sk}', s)$, $\quad \mathsf{pk} := \mathsf{pk}'$
- **return** $(\mathsf{pk}, \mathsf{sk})$

Alg $\mathsf{Encaps}(\mathsf{pk})$
- $\mathsf{m} \xleftarrow{\$} \{0,1\}^{\ell_m}$, $\quad r := \mathsf{G}(\mathsf{m})$
- $c := \mathsf{Enc}(\mathsf{pk}, \mathsf{m}; r)$, $\quad K := \mathsf{H}(\mathsf{m}, c)$
- **return** (K, c)

Alg $\mathsf{Decaps}(\mathsf{sk}, c)$
- **parse** $(\mathsf{sk}', s) := \mathsf{sk}$
- $\mathsf{m}' := \mathsf{Dec}(\mathsf{sk}', c)$
- **if** $\mathsf{m}' = \perp \vee \mathsf{Enc}(\mathsf{pk}, \mathsf{m}'; \mathsf{G}(\mathsf{m}')) \neq c$: $\quad$ **return** $K := \mathsf{H}(s, c)$
- **return** $K := \mathsf{H}(\mathsf{m}', c)$

Figure 2: The key encapsulation mechanism $\mathsf{KEM}[\mathsf{PKE}, \mathsf{H}, \mathsf{G}] = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ constructed by applying the Fujisaki-Okamoto transform [FO99, HHK17] to public key encryption scheme $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{Enc}, \mathsf{Dec})$ with randomness space $\{0,1\}^{\ell_r}$, and random oracles $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^{\ell_k}$, $\mathsf{G}\colon \{0,1\}^* \to \{0,1\}^{\ell_r}$.

3 Kleptographic Model

In this section, we precisely define what we call a kleptographic attack against implicit rejection. We state the syntax of such an attack and which properties it should have. To the best of our knowledge, no formal model for kleptography in general is known, but our model follows the main intuitions of previous models, namely, a kleptographic attack has to be *successful* and *undetectable*. Throughout the section, we fix a public key encryption scheme $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with public key space $\{0,1\}^{\ell_p}$, secret key space $\{0,1\}^{\ell_s}$, message space $\{0,1\}^{\ell_m}$, randomness space $\{0,1\}^{\ell_r}$, and ciphertext space $\{0,1\}^{\ell_c}$. We also fix parameters $\lambda, \ell_k \in \mathbb{N}$ and random oracles $\mathsf{G}\colon \{0,1\}^* \to \{0,1\}^{\ell_r}$ and $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^{\ell_k}$ as in the Fujisaki-Okamoto transform, see Section 2.

Let us first explain the syntax of an attack and how it is executed. The attack has four components: in a first phase, we allow the attack to perform some precomputation. This is modeled by an algorithm ASetup that outputs the attacker's public and secret key. In the second phase, the victim would run a subverted key generation algorithm of the Fujisaki-Okamoto transform, and potentially a subverted decapsulation algorithm. Precisely, we allow the attack to manipulate the way the seed s (see Gen in Figure 2) is derived in key generation, and we allow the attack to manipulate how implicit rejection keys K are derived (see Decaps in Figure 2). This derivation is modeled by algorithms $\mathsf{AGen}, \mathsf{ADecaps}$ and can depend on the attacker's public key and the secret key sk'. We may think of $\mathsf{AGen}, \mathsf{ADecaps}$ as the subverted code that is embedded on the victims machine. Notably, we do not allow the attacker to tamper with the parts of the code that are relevant during a correct execution of a key exchange, which ensures that the code of the victim remains fully functional. The online phase of the attack is modeled by an adversary $\mathsf{AOnline}$ which gets the attacker's secret key and has access to a decapsulation oracle.

Definition 5 (Attack Scheme)**.** An attack scheme is defined to be a quadruple $\mathsf{AS} = (\mathsf{ASetup}, \mathsf{AGen}, \mathsf{ADecaps}, \mathsf{AOnline})$ of algorithms with the following syntax:

- $\mathsf{ASetup} \to (\mathsf{apk}, \mathsf{ask})$ does not take any input and outputs an attacker public key apk and an attacker secret key ask.
- $\mathsf{AGen}(\mathsf{apk}, \mathsf{sk}') \to s$ takes as input an attacker public key apk and a secret key $\mathsf{sk}' \in \{0,1\}^{\ell_s}$ and outputs a seed $s \in \{0,1\}^{\lambda}$.
- $\mathsf{ADecaps}(\mathsf{apk}, \mathsf{sk}', s, c) \to K$ is deterministic, takes as input an attacker public key apk, a secret key $\mathsf{sk}' \in \{0,1\}^{\ell_s}$, a seed $s \in \{0,1\}^{\lambda}$, and a ciphertext $c \in \{0,1\}^{\ell_c}$, and outputs a key $K \in \{0,1\}^{\ell_k}$.
- $\mathsf{AOnline}^{\textsc{Dec},\mathsf{H},\mathsf{G}}(\mathsf{pk}, \mathsf{ask}) \to \mathsf{sk}'$ takes as input a public key $\mathsf{pk} \in \{0,1\}^{\ell_p}$ and an attacker secret key ask, has oracle access to a decapsulation oracle and random oracles, and outputs a secret key $\mathsf{sk}' \in \{0,1\}^{\ell_s}$.

Subsequently, we define the properties that an attack scheme should have. Obviously, we want that the attack is successful. To define that more precisely, we introduce the advantage of an attack, where we assume that the attack is run in a key-recovery game with respect to chosen-ciphertext attacks.

Definition 6 (Advantage of Attack Scheme)**.** Let $\mathsf{AS} = (\mathsf{ASetup}, \mathsf{AGen}, \mathsf{ADecaps}, \mathsf{AOnline})$ be an attack scheme. The advantage of AS is defined to be

$$\mathsf{Adv}^{\mathsf{SUB\text{-}KR\text{-}CCA}}_{\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} := \Pr\left[\mathbf{SUB\text{-}KR\text{-}CCA}^{\mathsf{AS}}_{\mathsf{PKE},\mathsf{H},\mathsf{G}} \Rightarrow 1\right],$$

where game **SUB-KR-CCA** is specified in Figure 3.

In addition to the attack being successful, we also want it to be undetectable. Precisely, we assume that the victim has black-box access to the potentially subverted key generation algorithm and decapsulation oracle of the Fujisaki-Okamoto transform. This is a reasonable assumption, as in practice keys are stored in secure modules and users only make functionality tests without getting direct access to the keys. In that case, we want that the outputs do not leak whether the algorithm has been subverted or not. There are different flavors of this property, and we define them next.

Definition 7 (Distinguishing Advantages)**.** Consider an attack scheme $\mathsf{AS} = (\mathsf{ASetup}, \mathsf{AGen}, \mathsf{ADecaps}, \mathsf{AOnline})$ and an algorithm $\mathcal{D}$. Let $\mathsf{Gen}^{\mathsf{AGen}(\mathsf{apk},\cdot)}$ and $\textsc{Dec}^{\mathsf{ADecaps}(\mathsf{apk},\cdot,\cdot,\cdot)}$ be as in Figure 3. We define the following advantages:

- **Key Pair.** The *key pair* distinguishing advantage of $\mathcal{D}$ against AS is defined to be

$$\mathsf{Adv}^{\mathsf{dist\text{-}kp}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} := \Bigg| \Pr\left[\mathcal{D}^{\mathsf{H},\mathsf{G}}(\mathsf{pk},\mathsf{sk}) = 1 \;\middle|\; (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KEM}[\mathsf{PKE},\mathsf{H},\mathsf{G}].\mathsf{Gen}\right] - \Pr\left[\mathcal{D}^{\mathsf{H},\mathsf{G}}(\mathsf{pk},\mathsf{sk}) = 1 \;\middle|\; \begin{array}{l}(\mathsf{apk},\mathsf{ask}) \leftarrow \mathsf{ASetup},\\ (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}^{\mathsf{AGen}(\mathsf{apk},\cdot)}\end{array}\right] \Bigg|.$$

Game $\textbf{SUB-KR-CCA}^{\mathsf{AS}}_{\mathsf{PKE,H,G}}$

$(\mathsf{apk}, \mathsf{ask}) \leftarrow \mathsf{ASetup}$
$(\mathsf{pk}, \mathsf{sk} = (\mathsf{sk}', s)) \leftarrow \mathsf{Gen}^{\mathsf{AGen}(\mathsf{apk},\cdot)}$
$\mathsf{sk}^* \leftarrow \mathsf{AOnline}^{\mathrm{DEC}^{\mathsf{ADecaps}(\mathsf{apk},\cdot,\cdot,\cdot)},\mathsf{H},\mathsf{G}}(\mathsf{pk}, \mathsf{ask})$
if $(\mathsf{pk}, \mathsf{sk}') \in \mathsf{PKE.Gen}$: **return** 1
return 0

Alg $\mathsf{Gen}^{\mathsf{AGen}(\mathsf{apk},\cdot)}$

$(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{PKE.Gen}$
$s \leftarrow \mathsf{AGen}(\mathsf{apk}, \mathsf{sk}')$
return $(\mathsf{pk} := \mathsf{pk}', \mathsf{sk} := (\mathsf{sk}', s))$

Oracle $\mathrm{DEC}^{\mathsf{ADecaps}(\mathsf{apk},\cdot,\cdot,\cdot)}(c)$

parse $(\mathsf{sk}', s) := \mathsf{sk}$
$m' := \mathsf{Dec}(\mathsf{sk}', c)$
if $m' = \bot \vee \mathsf{Enc}(\mathsf{pk}, m'; \mathsf{G}(m')) \neq c$:
  return $K := \mathsf{ADecaps}(\mathsf{apk}, \mathsf{sk}', s, c)$
return $K := \mathsf{H}(m', c)$

Figure 3: The subverted key-recovery game **SUB-KR-CCA** for an attack scheme $\mathsf{AS} = (\mathsf{ASetup}, \mathsf{AGen}, \mathsf{ADecaps}, \mathsf{AOnline})$. Algorithm $\mathsf{Gen}^{\mathsf{AGen}(\mathsf{apk},\cdot)}$ models a subverted key generation procedure for the Fujisaki-Okamoto transform presented in Figure 2 and oracle $\mathrm{DEC}^{\mathsf{ADecaps}(\mathsf{apk},\cdot,\cdot,\cdot)}$ models a subverted decapsulation algorithm. The highlighted lines are the only change compared to the benign algorithms in Figure 2.

- **Oracle.** The *oracle* distinguishing advantage of $\mathcal{D}$ against AS is defined to be

$$\mathsf{Adv}^{\mathsf{dist\text{-}o}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} := \left| \Pr\left[\mathcal{D}^{\mathsf{Decaps}(\mathsf{sk},\cdot),\mathsf{H},\mathsf{G}}(\mathsf{pk}, \mathsf{sk}') = 1 \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KEM}[\mathsf{PKE}, \mathsf{H}, \mathsf{G}].\mathsf{Gen}, \\ (\mathsf{sk}', s) := \mathsf{sk} \end{array} \right] - \Pr\left[\mathcal{D}^{\mathrm{DEC}^{\mathsf{ADecaps}(\mathsf{apk},\cdot,\cdot,\cdot)},\mathsf{H},\mathsf{G}}(\mathsf{pk}, \mathsf{sk}') = 1 \;\middle|\; \begin{array}{l} (\mathsf{apk}, \mathsf{ask}) \leftarrow \mathsf{ASetup}, \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}^{\mathsf{AGen}(\mathsf{apk},\cdot)}, \\ (\mathsf{sk}', s) := \mathsf{sk} \end{array} \right] \right|,$$

- **Attacker Key Oracle.** The *attacker key oracle* distinguishing advantage of $\mathcal{D}$ against AS is defined to be

$$\mathsf{Adv}^{\mathsf{dist\text{-}ako}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} := \left| \Pr\left[\mathcal{D}^{\mathsf{Decaps}(\mathsf{sk},\cdot),\mathsf{H},\mathsf{G}}(\mathsf{apk}, \mathsf{pk}, \mathsf{sk}') = 1 \;\middle|\; \begin{array}{l} (\mathsf{apk}, \mathsf{ask}) \leftarrow \mathsf{ASetup}, \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KEM}[\mathsf{PKE}, \mathsf{H}, \mathsf{G}].\mathsf{Gen}, \\ (\mathsf{sk}', s) := \mathsf{sk} \end{array} \right] - \Pr\left[\mathcal{D}^{\mathrm{DEC}^{\mathsf{ADecaps}(\mathsf{apk},\cdot,\cdot,\cdot)},\mathsf{H},\mathsf{G}}(\mathsf{apk}, \mathsf{pk}, \mathsf{sk}') = 1 \;\middle|\; \begin{array}{l} (\mathsf{apk}, \mathsf{ask}) \leftarrow \mathsf{ASetup}, \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}^{\mathsf{AGen}(\mathsf{apk},\cdot)}, \\ (\mathsf{sk}', s) := \mathsf{sk} \end{array} \right] \right|,$$

We shall elaborate on the motivation for defining precisely these advantages. Assuming that the distinguisher does not take decapsulation into account, in the definition of the key pair advantage we assume that the distinguisher treats the key generation algorithm as a black-box and does not investigate the decapsulation algorithm at all. In the definition of oracle advantage we additionally allow the distinguisher to have black-box access to a potentially subverted decapsulation oracle. Notably, if we were to give the distinguisher both the full secret key (including s) and black-box access to the decapsulation oracle, then it would be trivial to detect any subversion in the decapsulation oracle: the distinguisher would simply submit an invalid ciphertext to the oracle and compare the result with a local execution of honest decapsulation. The notion of attacker key oracle advantage additionally gives the distinguisher access to the attacker public key. One can see that if the subversion is indistinguishable in that model, we get a form of security preservation: even when a user learns the attacker's public key apk, e.g., by reverse-engineering the subverted algorithm, this user can not break the security (e.g., IND-CCA security) of other users who also use the subverted key generation algorithm. In other words, only the attacker itself can perform the attack. To see this, consider an IND-CCA game in which an adversary tries to break security of a user with subverted code, while getting not only the public key, but also the attacker's public key apk. Then, to argue that this adversary can not break IND-CCA, we can first use the undetectability notion (as in attacker key oracle) the switch to a hybrid experiment in which the code is not subverted. By standard IND-CCA security, the adversary can not win this game.

Remark 1 (Trivial Relations). It is clear that if key generation is not subverted, then the two experiments in the *key pair* setting are the same, and hence the advantage of each distinguisher is zero. Similarly, if decapsulation is not subverted, then the two experiments in the *oracle* setting are the same, hence the advantage of each distinguisher is zero. Also, in general, the advantage in the *attacker key oracle* setting is an upper bound for the advantage in the *oracle* setting, as a distinguisher in the *attacker key oracle* setting can simply ignore its additional input.

4 Our Attacks

Here, we present our kleptographic attacks against implicit rejection. The attacks follow the model defined in Section 3. Concretely, we present three attacks. The first attack subverts only the implicit rejection branch of decapsulation, is very efficient, and is in the public key setting, i.e., reverse-engineering the subverted algorithm does not help to carry out the attack. The second and third attack subvert key generation only and do not tamper with decapsulation. They are less efficient and are in the secret key setting, i.e., the attacker has to embed its secret key in the subverted key generation algorithm. What distinguishes these two attacks is that the third attack shows a time-space trade-off using preprocessing.

Attack Target. For all attacks, we again fix a scheme as in the Fujisaki-Okamoto transform. That is, we let $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme with public key space $\{0,1\}^{\ell_p}$, secret key space $\{0,1\}^{\ell_s}$, message space $\{0,1\}^{\ell_m}$, randomness space $\{0,1\}^{\ell_r}$, and ciphertext space $\{0,1\}^{\ell_c}$. We assume parameters $\lambda, \ell_k \in \mathbb{N}$ and random oracles $\mathsf{G}\colon \{0,1\}^* \to \{0,1\}^{\ell_r}$ and $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^{\ell_k}$. Our attacks target the scheme $\mathsf{KEM}[\mathsf{PKE}, \mathsf{H}, \mathsf{G}]$, see Figure 2.

Kyber as a Running Example. To illustrate the concrete efficiency and advantage of our attacks, we will use Kyber [ABD+21] with parameter set Kyber1024 as a running example. According to the specification, the relevant paramters are $\ell_s = \lambda = \ell_k = 32 \cdot 8 = 256$ and $\ell_c = 1568 \cdot 8 = 12544$. Note that we do not assume that the secret key is a short vector, but rather the seed to generate it. Throughout, we make the assumption that the spreadness of Kyber is $\gamma \leq 1/1000$. This is a reasonable assumption, as large γ would clearly lead to security issues.

4.1 Subverting Decapsulation Only

In our first attack, we tamper with decapsulation but not with key generation. The main idea is to let the implicit rejection keys K output by decapsulation on failure be encryptions of the secret key sk'. In this way, we establish a hidden channel of communication via which the attacker can extract sk'. Making this idea work comes with subtle challenges we have to overcome. First, we can not just assume that the key K is large enough to hold an entire encryption of sk'. To solve this, we further split the encryption into chunks, and address the chunks using the first bits of the ciphertext. Second, we need to ensure that K is deterministically derived from the ciphertext. A first approach is to use the remaining part of the ciphertext as the randomness for the encryption. In this case, however, we can not argue that the input-output behavior of the subverted decapsulation is indistinguishable, as the distinguisher would have full control over the randomness used for encryption. The solution is to use the seed s to derive a pseudorandom string from the ciphertext, and then use this string as the randomness for encryption. Although s is not subverted, it is assumed to have enough entropy and is hidden from the distinguisher.

Attack Description. Let $\widehat{\mathsf{PKE}} = (\widehat{\mathsf{Gen}}, \widehat{\mathsf{Enc}}, \widehat{\mathsf{Dec}})$ be a public key encryption scheme with public key space $\{0,1\}^{\hat{\ell}_p}$, secret key space $\{0,1\}^{\hat{\ell}_s}$, message space $\{0,1\}^{\hat{\ell}_m}$, randomness space $\{0,1\}^{\hat{\ell}_r}$, and ciphertext space $\{0,1\}^{\hat{\ell}_c}$. The only requirement for the relation of these parameters of $\widehat{\mathsf{PKE}}$ to the parameters of the target scheme $\mathsf{KEM}[\mathsf{PKE}, \mathsf{H}, \mathsf{G}]$ is that $\hat{\ell}_m \geq \ell_s$, i.e., $\widehat{\mathsf{PKE}}$ can encrypt sk'. We assume that $\widehat{\mathsf{PKE}}$ is perfecly correct, i.e., it has correctness error $\delta = 0$. Further, let $\hat{\mathsf{H}}\colon \{0,1\}^* \to \{0,1\}^{\hat{\ell}_r}$ be a random oracle. We give a formal presentation of our attack in Figure 4.

Attack Analysis. We formally analyze our attack, thereby showing that it is (1) efficient, (2) successful and (3) undetectable. For (1), efficiency of our attack follows easily by inspection. For (2), we analyze the advantage of our attack according to Definition 6. For (3), we bound the distinguishing advantages of the attack according to Definition 7.

Alg ASetup

$(\mathsf{apk}, \mathsf{ask}) \leftarrow \widehat{\mathsf{Gen}}$
return $(\mathsf{apk}, \mathsf{ask})$

Alg $\mathsf{ADecaps}(\mathsf{apk}, \mathsf{sk}', s, c)$

parse $(\mathsf{ind}, \mathsf{rnd}) := c,\ \mathsf{ind} \in \{0,1\}^t,\ \mathsf{rnd} \in \{0,1\}^{\ell_c - t}$
$\hat{K} := \widehat{\mathsf{Enc}}(\mathsf{apk}, \mathsf{sk}'; \hat{\mathsf{H}}(\mathsf{rnd}, s))$
parse $(\hat{K}_1, \ldots, \hat{K}_T) := \hat{K} \in (\{0,1\}^{\ell_k})^T$
parse $i := \mathsf{ind},\ i \in [T]$
return $K := \hat{K}_i$

Alg $\mathsf{AGen}(\mathsf{apk}, \mathsf{sk}')$

return $s \xleftarrow{\$} \{0,1\}^\lambda$

Alg $\mathsf{AOnline}^{\textsc{Dec},\mathsf{H},\mathsf{G}}(\mathsf{pk}, \mathsf{ask})$

$\mathsf{rnd} \xleftarrow{\$} \{0,1\}^{\ell_c - t}$
for $i \in [T]$:
  parse $\mathsf{ind} := i,\ \mathsf{ind} \in \{0,1\}^t$
  $c_i := (\mathsf{ind}, \mathsf{rnd}) \in \{0,1\}^{\ell_c}$
  $\hat{K}_i := \textsc{Dec}(c_i)$
$\hat{K} := (\hat{K}_1, \ldots, \hat{K}_T) \in \{0,1\}^{\hat{\ell}_c}$
return $\mathsf{sk}' := \widehat{\mathsf{Dec}}(\mathsf{ask}, \hat{K})$

Figure 4: Attack scheme $\mathsf{AS} = (\mathsf{ASetup}, \mathsf{AGen}, \mathsf{ADecaps}, \mathsf{AOnline})$ subverting only the decapsulation algorithm. We have $T := \lceil \hat{\ell}_c / \ell_k \rceil$ and $t := \lceil \log(T) \rceil$. Further, $\hat{\mathsf{H}} \colon \{0,1\}^* \to \{0,1\}^{\hat{\ell}_r}$ is a random oracle and we assume $\hat{\ell}_m \geq \ell_s$.

Lemma 1 (Online Complexity)**.** *Consider the attack scheme* AS *in Figure 4. Then, algorithm* $\mathsf{AOnline}$ *issues at most* $T = \lceil \hat{\ell}_c / \ell_k \rceil$ *decapsulation queries, and does not compute any hash evaluation.*

Proof. This follows easily by inspection. □

Lemma 2 (Advantage)**.** *Assume that* PKE *has spreadness* $\gamma \in [0,1]$ *and consider the attack scheme* AS *in Figure 4. Then, we have*

$$\mathsf{Adv}^{\mathsf{SUB\text{-}KR\text{-}CCA}}_{\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} \geq 1 - T \cdot \gamma.$$

Proof. It is easy to verify that the attack succeeds as long as all queries to the decapsulation oracle trigger implicit rejection, i.e., if on every query $\textsc{Dec}(c_i)$ that $\mathsf{AOnline}^{\textsc{Dec},\mathsf{H},\mathsf{G}}(\mathsf{pk}, \mathsf{ask})$ issues, $\textsc{Dec}$ internally calls $\mathsf{ADecaps}(\mathsf{apk}, \mathsf{sk}', s, c)$. In other words, the attack succeeds if the following event does not occur, where the probability space is defined by random oracle G, the keys $(\mathsf{pk}, \mathsf{sk}') \leftarrow \mathsf{Gen}$, and the randomness $\mathsf{rnd} \xleftarrow{\$} \{0,1\}^{\ell_c - t}$:

- Event Valid: This event occurs, if there is an $i \in [T]$ such that for $c_i = (\mathsf{ind}, \mathsf{rnd}) \in \{0,1\}^{\ell_c}$ where $\mathsf{ind} \in \{0,1\}^t$ is the binary representation of i and $\mathsf{m}' := \mathsf{Dec}(\mathsf{sk}', c_i)$ we have $\mathsf{m}' \neq \bot$ and $\mathsf{Enc}(\mathsf{pk}, \mathsf{m}'; \mathsf{G}(\mathsf{m}')) = c_i$.

To bound the probability of event Valid, we use a union bound over all $i \in [T]$. Denote by Valid_i the event that Valid occurs for a specific $i \in [T]$. The probability of Valid_i is easily bounded by the spreadness γ of PKE. More formally, if we fix c_i, we have

$$\begin{aligned}
\Pr_{\mathsf{pk},\mathsf{sk}',\mathsf{G}}[\mathsf{Valid}_i] &\leq \Pr_{\mathsf{pk},\mathsf{sk}',\mathsf{G}}[\mathsf{Enc}(\mathsf{pk}, \mathsf{m}'; \mathsf{G}(\mathsf{m}')) = c_i \mid \mathsf{m}' \neq \bot] \\
&= \Pr_{\mathsf{pk},\mathsf{sk}',\rho}[\mathsf{Enc}(\mathsf{pk}, \mathsf{m}'; \rho) = c_i \mid \mathsf{m}' \neq \bot] \\
&= \mathop{\mathbb{E}}_{\mathsf{pk},\mathsf{sk}'}\left[\Pr_{\rho}[\mathsf{Enc}(\mathsf{pk}, \mathsf{m}'; \rho) = c_i \mid \mathsf{m}' \neq \bot]\right] \\
&\leq \mathop{\mathbb{E}}_{\mathsf{pk},\mathsf{sk}'}\left[\max_{\mathsf{m},c} \Pr_{\rho}[\mathsf{Enc}(\mathsf{pk}, \mathsf{m}; \rho) = c]\right] \leq \gamma.
\end{aligned}$$

In combination, we get

$$\mathsf{Adv}^{\mathsf{SUB\text{-}KR\text{-}CCA}}_{\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} \geq 1 - \Pr[\mathsf{Valid}] \geq 1 - T \cdot \gamma.$$

□

Lemma 3 (Undetectability)**.** *Consider the attack scheme* AS *in Figure 4. Then, for any algorithm* $\mathcal{D}$ *that makes at most* Q *queries to the random oracles* $\mathsf{H}, \mathsf{G}, \hat{\mathsf{H}}$ *in total and at most* Q_D *queries to its* *decapsulation oracle, there is an algorithm $\mathcal{D}'$ with $\mathbf{T}(\mathcal{D}) \approx \mathbf{T}(\mathcal{D}')$ such that*

$$\mathsf{Adv}^{\mathsf{dist\text{-}kp}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} = 0, \quad \mathsf{Adv}^{\mathsf{dist\text{-}o}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} \leq \mathsf{Adv}^{\mathsf{dist\text{-}ako}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} \leq \frac{2Q}{2^\lambda} + Q_D \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA\text{-}R}}_{\mathcal{D}',\widehat{\mathsf{PKE}}}.$$

Proof. First, because the attack we consider does not subvert key generation at all, it is clear that

$$\mathsf{Adv}^{\mathsf{dist\text{-}kp}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} = 0.$$

Further, the inequality $\mathsf{Adv}^{\mathsf{dist\text{-}o}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} \leq \mathsf{Adv}^{\mathsf{dist\text{-}ako}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}}$ always holds. Therefore, we only need to bound the advantage of $\mathcal{D}$ in the *attacker key oracle* setting. To do so, we present a sequence of games $\mathbf{G}_0$ to $\mathbf{G}_5$, where $\mathbf{G}_0$ and $\mathbf{G}_5$ correspond to the games that $\mathcal{D}$ has to distinguish in the *attacker key oracle* setting. We will have

$$\mathsf{Adv}^{\mathsf{dist\text{-}ako}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} = \left|\Pr\left[\mathbf{G}_0 \Rightarrow 1\right] - \Pr\left[\mathbf{G}_5 \Rightarrow 1\right]\right|.$$

Game $\mathbf{G}_0$: In $\mathbf{G}_0$, $\mathcal{D}$ gets as input an attacker public key apk, and the pair $(\mathsf{pk}, \mathsf{sk}')$ where $(\mathsf{apk}, \mathsf{ask}) \leftarrow \widehat{\mathsf{Gen}}$ is the attacker's key pair and $(\mathsf{pk}, \mathsf{sk}') \leftarrow \mathsf{PKE.Gen}(1^\lambda)$. The game additionally samples the seed $s \xleftarrow{\$} \{0,1\}^\lambda$ which is hidden from $\mathcal{D}$. The distinguisher $\mathcal{D}$ also gets access to random oracles H, G, and $\hat{\mathsf{H}}$, which are implemented using standard lazy sampling. Additionally, it gets access to a subverted decapsulation oracle $\textsc{Dec}^{\mathsf{ADecaps}(\mathsf{apk},\cdot,\cdot,\cdot)}$. In the next games, it will be our goal to replace this subverted oracle with the honest oracle. To recall the difference, in the honest oracle, an implicit rejection key for ciphertext c is $K := \mathsf{H}(s, c)$, whereas it is computed as in $\mathsf{ADecaps}(\mathsf{apk}, \mathsf{sk}', s, c)$ (see Figure 4) in the subverted oracle.

Game $\mathbf{G}_1$: In this game, we let the game terminate and output 0 whenever one of the following two events occurs:

- Event QryA : This event occurs, if $\mathcal{D}$ ever directly queries $\hat{\mathsf{H}}(\mathsf{rnd}, s)$ for some $\mathsf{rnd} \in \{0,1\}^{\ell_c - t}$.
- Event QryB : This event occurs, if $\mathcal{D}$ ever directly queries $\mathsf{H}(s, c)$ for some c.

As s is uniform over $\{0,1\}^\lambda$ and hidden from $\mathcal{D}$, for each fixed random oracle query, the probability that QryA or QryB occurs is at most $1/2^\lambda$. With a union bound over all queries, we get

$$\left|\Pr\left[\mathbf{G}_0 \Rightarrow 1\right] - \Pr\left[\mathbf{G}_1 \Rightarrow 1\right]\right| \leq \Pr\left[\mathsf{QryA} \vee \mathsf{QryB}\right] \leq \frac{Q}{2^\lambda}.$$

Game $\mathbf{G}_2$: Recall that in $\mathbf{G}_1$, when algorithm $\mathsf{ADecaps}(\mathsf{apk}, \mathsf{sk}', s, c)$ is called during a query to the decapsulation oracle, then the algorithm computes $\hat{K} := \widehat{\mathsf{Enc}}(\mathsf{apk}, \mathsf{sk}'; \hat{\mathsf{H}}(\mathsf{rnd}, s))$. In $\mathbf{G}_2$, we change this to $\hat{K} := \mathsf{HK}[\mathsf{rnd}]$, where $\mathsf{HK}[\cdot]$ is a map that lazily implements a random function $\{0,1\}^{\ell_c - t} \to \{0,1\}^{\hat{\ell}_c}$. As we can assume that QryA does not occur, each $\hat{K}$ in $\mathbf{G}_1$ looks like a ciphertext under $\widehat{\mathsf{PKE}}$ computed with uniform random coins. Therefore, we can use the $\mathsf{IND\text{-}CPA\text{-}R}$ security of $\widehat{\mathsf{PKE}}$ in Q_D hybrid steps as follows. We define $\mathbf{G}_{1,i}$, which is as $\mathbf{G}_1$, but for the first i strings rnd that are submitted to the decapsulation oracle (as part of c), $\mathsf{ADecaps}(\mathsf{apk}, \mathsf{sk}', s, c)$ behaves as in game $\mathbf{G}_2$, whereas all remaining ones are as in $\mathbf{G}_1$. Note that this means if a specific rnd is submitted twice and $\hat{K}$ has to be computed twice, then the game computes the same $\hat{K}$ in both invocations. Then, to argue that $\mathbf{G}_{1,i}$ and $\mathbf{G}_{1,i+1}$ are indistinguishable we build a reduction $\mathcal{D}'$ which runs in the $\mathsf{IND\text{-}CPA\text{-}R}$ game of $\widehat{\mathsf{PKE}}$. It gets as input apk and simulates the game $\mathbf{G}_{1,i}$ for $\mathcal{D}$, except for the $i+1$st string rnd that is submitted to the decapsulation oracle. For this string, if $\hat{K}$ has to be computed, it submits sk' to the $\mathsf{IND\text{-}CPA\text{-}R}$ game and gets $\hat{K}$ (a ciphertext with respect to $\widehat{\mathsf{PKE}}$) back. Finally, it outputs whatever the game outputs. With this hybrid argument, we get

$$\left|\Pr\left[\mathbf{G}_1 \Rightarrow 1\right] - \Pr\left[\mathbf{G}_2 \Rightarrow 1\right]\right| \leq Q_D \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA\text{-}R}}_{\mathcal{D}',\widehat{\mathsf{PKE}}}.$$

Game $\mathbf{G}_3$: In this game, we change how the chunks $\hat{K}_i$ of $\hat{K}$ are computed. Recall that in $\mathbf{G}_2$, when algorithm $\mathsf{ADecaps}(\mathsf{apk}, \mathsf{sk}', s, c)$ is called during a query to the decapsulation oracle, then the algorithm computes $\hat{K} := \mathsf{HK}[\mathsf{rnd}]$ and then splits it into T chunks $\hat{K}_1, \ldots, \hat{K}_T$. In $\mathbf{G}_3$, the game no longer computes $\hat{K}$ via a HK, but instead computes each chunk $\hat{K}_i$ as $\hat{K}_i := \mathsf{HK}'[\mathsf{ind}, \mathsf{rnd}]$, where ind

is the binary representation of $i \in [T]$ and $\mathsf{HK}'[\cdot,\cdot]$ is a map that lazily implements a random function $\{0,1\}^t \times \{0,1\}^{\ell_c - t} \to \{0,1\}^{\ell_k}$. It is clear that this change is only conceptual, and we have

$$\Pr[\mathbf{G}_2 \Rightarrow 1] = \Pr[\mathbf{G}_3 \Rightarrow 1].$$

Game $\mathbf{G}_4$: In this game, whenever $\mathsf{HK}'[\mathsf{ind},\mathsf{rnd}]$ has to sampled, the game assembles $c = (\mathsf{ind},\mathsf{rnd})$ and defines $\mathsf{HK}'[\mathsf{ind},\mathsf{rnd}] := \mathsf{H}(s,c)$. As we assume that QryB does not occur, we have

$$\Pr[\mathbf{G}_3 \Rightarrow 1] = \Pr[\mathbf{G}_4 \Rightarrow 1].$$

Game $\mathbf{G}_5$: In this game, we no longer output 0 if QryA or QryB occurs. With arguments as in $\mathbf{G}_1$, we get

$$|\Pr[\mathbf{G}_4 \Rightarrow 1] - \Pr[\mathbf{G}_5 \Rightarrow 1]| \leq \Pr[\mathsf{QryA} \vee \mathsf{QryB}] \leq \frac{Q}{2^\lambda}.$$

Now, one can easily observe that $\mathbf{G}_5$ corresponds to the game with the honest decapsulation oracle, finishing the proof. □

Concrete Example: Kyber. To recall, the secret key in Kyber has length $\ell_s = 256$ and encapsulated keys have length $\ell_k = 256$. This means we can use (hashed) ElGamal [ElG84, ABR98] with SHA-256 to instantiate $\widehat{\mathsf{PKE}}$. Instantiating ElGamal over a standard 256-bit group, we obtain $\hat{\ell}_c < 3 \cdot \ell_k$, i.e., $T = 3$ decapsulation queries are sufficient to make the attack work. We get an advantage of at least $1 - 3/1000$.

4.2 Subverting Key Generation Only

The attack we present in this section does not tamper with decapsulation. That is, the key K on implicit rejection is computed as in the Fujisaki-Okamoto transform (see Figure 2) from the ciphertext c and the seed s. However, instead of sampling s uniformly, we subvert key generation to make s depend on the primary secret key sk'. In this way, the attacker can gain information about sk' from rejection keys K. More precisely, let $\hat{\mathsf{H}}\colon \{0,1\}^* \to \{0,1\}^\lambda$ be a random oracle. The subverted key generation sets $s := \hat{\mathsf{H}}(\mathsf{apk}, \sigma)$, where σ are the first $h \leq \ell_s$ bits of sk' and $\mathsf{apk} \in \{0,1\}^\lambda$ is the random key embedded by the attacker. Clearly, s remains pseudorandom as long as apk is not leaked. At the same time, the attacker can make one decapsulation query to get an implicit rejection key K and then do an exhaustive search to find σ. In a second step, the attacker can do another exhaustive search over the remaining bits of sk'. Setting $h = \ell_s/2$ yields an attack of complexity roughly $2^{\ell_s/2+1}$.

Attack Description. We present our attack in Figure 5, where we assume that one can efficiently check if $(\mathsf{pk},\mathsf{sk}') \in \mathsf{Gen}$ for given $(\mathsf{pk},\mathsf{sk}')$. This assumption holds for most natural schemes, especially if we assume pk to be derived from sk', for example if sk' is a seed for a pseudorandom generator from which the actual key pair is derived.

Attack Analysis. Below, we show that our attack successful and undetectable, and analyze its efficiency. That is, we count the number of operations, and analyze the advantage according to Definition 6 and the distinguishing advantages according to Definition 7. Note that the attack is easily detectable once apk is leaked, and anyone holding apk can perform the attack.

Lemma 4 (Online Complexity)**.** *Consider the attack scheme* AS *in Figure 5. Then, algorithm* $\mathsf{AOnline}$ *issues at most one decapsulation query, computes at most* 2^{h+1} *hash evaluations, and checks* $(\mathsf{pk},\mathsf{sk}') \in \mathsf{Gen}$ *at most* $2^{\ell_s - h}$ *times.*

Proof. The lemma follows easily by inspection: during the online phase, the attack issues one decapsulation query and computes at most $2^h \cdot 2 = 2^{h+1}$ hashes in algorithm $\mathsf{PrefixSearch}$. Further, it checks key pairs in algorithm $\mathsf{FinalSearch}$ at most $2^{\ell_s - h}$ times. □

Lemma 5 (Advantage)**.** *Assume that* PKE *has spreadness* $\gamma \in [0,1]$ *and consider the attack scheme* AS *in Figure 5. Then, we have*

$$\mathsf{Adv}^{\mathsf{SUB\text{-}KR\text{-}CCA}}_{\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} \geq 1 - (\gamma + 2^{-\lambda} + 2^{-\ell_k}).$$

Alg ASetup

$\mathsf{apk} \xleftarrow{\$} \{0,1\}^{\lambda}$
return $(\mathsf{apk}, \mathsf{ask} := \mathsf{apk})$

Alg $\mathsf{ADecaps}(\mathsf{apk}, \mathsf{sk}', s, c)$

return $K := \mathsf{H}(s, c)$

Alg $\mathsf{PrefixSearch}(\mathsf{apk})$

$c^* \xleftarrow{\$} \{0,1\}^{\ell_c}$
$K^* := \textsc{Dec}(c^*)$
for $\sigma \in \{0,1\}^h$:
  $s := \hat{\mathsf{H}}(\mathsf{apk}, \sigma)$
  if $K^* = \mathsf{H}(s, c^*)$: **return** σ
return $\perp$

Alg $\mathsf{AGen}(\mathsf{apk}, \mathsf{sk}')$

parse $(\sigma, \sigma') := \mathsf{sk}', \ \sigma \in \{0,1\}^h$
return $s := \hat{\mathsf{H}}(\mathsf{apk}, \sigma)$

Alg $\mathsf{FinalSearch}(\mathsf{pk}, \sigma)$

for $\sigma' \in \{0,1\}^{\ell_s - h}$:
  $\mathsf{sk}' := (\sigma, \sigma') \in \{0,1\}^{\ell_s}$
  if $(\mathsf{pk}, \mathsf{sk}') \in \mathsf{Gen}$: **return** sk'
return $\perp$

Alg $\mathsf{AOnline}^{\textsc{Dec}, \mathsf{H}, \mathsf{G}, \hat{\mathsf{H}}}(\mathsf{pk}, \mathsf{ask})$

parse $\mathsf{apk} := \mathsf{ask}$
$\sigma \leftarrow \mathsf{PrefixSearch}(\mathsf{apk})$
return $\mathsf{sk}' := \mathsf{FinalSearch}(\mathsf{pk}, \sigma)$

Figure 5: Attack scheme $\mathsf{AS} = (\mathsf{ASetup}, \mathsf{AGen}, \mathsf{ADecaps}, \mathsf{AOnline})$ and helper algorithms $\mathsf{PrefixSearch}$, $\mathsf{FinalSearch}$ subverting only the key generation. Here, $\hat{\mathsf{H}}\colon \{0,1\}^* \to \{0,1\}^{\lambda}$ is a random oracle. The attack has a parameter $h \leq \ell_s$ that influences its efficiency.

Proof. Recall that the attack first runs $\sigma \leftarrow \mathsf{PrefixSearch}(\mathsf{apk})$, where $\mathsf{PrefixSearch}$ internally samples a $c^* \xleftarrow{\$} \{0,1\}^{\ell_c}$, queries $K^* := \textsc{Dec}(c^*)$, and then iterates over all $\sigma \in \{0,1\}^h$, computes $s := \hat{\mathsf{H}}(\mathsf{apk}, \sigma)$ and outputs σ if $K^* = \mathsf{H}(s, c^*)$. Then, if σ is indeed the h-bit prefix of the secret key sk', then algorithm $\mathsf{FinalSearch}$ will find sk' and the attack succeeds. For our analysis, we denote the h-bit prefix of sk' by σ^* and define the following events:

- Event Find: This event occurs, if algorithm $\mathsf{PrefixSearch}(\mathsf{apk})$ outputs σ^*.
- Event Valid: This event occurs, if the ciphertext c^* sampled in algorithm $\mathsf{PrefixSearch}$ does not lead to implicit rejection, i.e., if re is an $i \in [T]$ such that for $\mathsf{m}' := \mathsf{Dec}(\mathsf{sk}', c^*)$ we have $\mathsf{m}' \neq \perp$ and $\mathsf{Enc}(\mathsf{pk}, \mathsf{m}'; \mathsf{G}(\mathsf{m}')) = c^*$.
- Event Coll: This event occurs, if there exists $\sigma' \in \{0,1\}^h \setminus \{\sigma^*\}$ such that $K^* = \mathsf{H}(\hat{\mathsf{H}}(\mathsf{apk}, \sigma'), c^*)$.

By our discussion above, it is clear that

$$\mathsf{Adv}^{\mathsf{SUB\text{-}KR\text{-}CCA}}_{\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} \geq 1 - \Pr[\neg\mathsf{Find}].$$

Now, as in the proof of Lemma 2, we can argue that the probability of event Valid is at most γ. Further, we can argue that the probability of Coll is at most $1/2^{\lambda}$ (for the case that $\hat{\mathsf{H}}(\mathsf{apk}, \sigma) = \hat{\mathsf{H}}(\mathsf{apk}, \sigma^*)$) plus $1/2^{\ell_k}$ (for the case that $\hat{\mathsf{H}}(\mathsf{apk}, \sigma) \neq \hat{\mathsf{H}}(\mathsf{apk}, \sigma^*)$ but $K^* = \mathsf{H}(\hat{\mathsf{H}}(\mathsf{apk}, \sigma'), c^*)$). Hence,

$$\Pr[\neg\mathsf{Find}] \leq \gamma + 2^{-\lambda} + 2^{-\ell_k} + \Pr[\neg\mathsf{Find} \mid \neg\mathsf{Valid} \wedge \neg\mathsf{Coll}].$$

Now, observe that if $\neg\mathsf{Valid}$, then when the loop in algorithm $\mathsf{PrefixSearch}$ considers σ^*, it will output σ^*. Hence, conditioned on $\neg\mathsf{Valid}$, the only way that $\neg\mathsf{Find}$ can happen is if a different σ' is output such that $K{*} = \mathsf{H}(\hat{\mathsf{H}}(\mathsf{apk}, \sigma'), c^*)$, which means that Coll occurs. So, we get

$$\Pr[\neg\mathsf{Find} \mid \neg\mathsf{Valid} \wedge \neg\mathsf{Coll}] = 0.$$

□

Lemma 6 (Undetectability)**.** *Consider the attack scheme* AS *in Figure 5. Then, for any algorithm* $\mathcal{D}$ *that makes at most* Q *queries to random oracle* $\hat{\mathsf{H}}$*, we have*

$$\mathsf{Adv}^{\mathsf{dist\text{-}kp}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} \leq \frac{Q}{2^{\lambda}}, \quad \mathsf{Adv}^{\mathsf{dist\text{-}o}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} = 0.$$

Proof. First, because decapsulation is not subverted, it is clear that the advantage in the *oracle* setting is zero. To bound the advantage in the *key pair* setting, let $\mathbf{G}$ denote the game with the honestly generated key pair and $\mathbf{G}'$ denote the game with the subverted key generation. We define the following event

- Event Qry: This event occurs, if $\mathcal{D}$ ever queries $\hat{\mathsf{H}}(\mathsf{apk}, \sigma)$ for some σ.

The probability that Qry occurs (in $\mathbf{G}$ and $\mathbf{G}'$) is at most $1/2^\lambda$ per random oracle query as apk is hidden from $\mathcal{D}$. By a union bound over all queries, the probability of Qry is at most $Q/2^\lambda$. Further, conditioned on $\neg\mathsf{Qry}$, it is clear that the view of $\mathcal{D}$ is the same in both games. Thus, we have

$$\left|\Pr\left[\mathbf{G} \Rightarrow 1\right] - \Pr\left[\mathbf{G}' \Rightarrow 1\right]\right| \leq \Pr\left[\mathsf{Qry}\right] \leq \frac{Q}{2^\lambda}.$$

□

Concrete Example: Kyber. As we have already mentioned, we would set $h = \ell_s/2$ to balance the complexity of the two exhaustive searches. For Kyber, this means $h = \ell_s/2 = 128$, leading to a complexity of $2^{128+1} + 2^{256-128} < 2^{130}$. The advantage is almost 1.

4.3 An Attack with Preprocessing

We generalize our attack from Section 4.2 by showing a time-memory trade-off. Key generation is subverted as in the attack in Section 4.2, i.e., we have $s = \hat{\mathsf{H}}(\mathsf{apk}, \sigma)$, where $\hat{\mathsf{H}}\colon \{0,1\}^* \to \{0,1\}^\lambda$ is a random oracle, σ are the first h bits of sk', and apk is a key embedded by the attacker. Decapsulation is not subverted. Looking back at our attack from Section 4.2, we were setting $h = \ell_s/2$, leading to two exhaustive searches over $\ell_s/2$ bits. The idea of the attack in this section is to speed up the first search that finds σ, which will allow us to increase h, thereby reducing the cost of the second search.

Preparation: Hellman Tables. We first recall the time-memory trade-off introduced by Hellman [Hel80], adapted to our setting. Thereby, we also introduce notation that we will then use in our attack description. In the time-memory trade-off for inverting a function F, we first do a preprocessing where we set up a table Tab with H rows and W columns. Roughly, each row i consists of a chain of evaluations of F. Namely, it contains the elements $z_i, \mathsf{F}(z_i), \mathsf{F}(\mathsf{F}(z_i)), \ldots, \mathsf{F}^W(z_i)$. The table, however, stores only the starting point z_i and the end point $\mathsf{F}^W(z_i)$ for each row. In the online phase, given y, we search for y in our list of endpoints $\mathsf{F}^W(z_1), \ldots, \mathsf{F}^W(z_H)$ (e.g., using binary search). If we find it, say in row i^*, we can recompute a preimage of y by computing $\mathsf{F}^{W-1}(z_{i^*})$. If not, we repeat the process for $\mathsf{F}(y)$, and so on. To increase the success probability, multiple independent variations of F have to be created and the approach has to be repeated using T such tables in parallel. In our situation, we can simply use c to introduce variations. Namely, we want to invert the function mapping $\sigma \in \{0,1\}^h$ to $K \in \{0,1\}^{\ell_k}$, or more specifically, mapping to the first h bits of K, where we assume[1] $h \leq \ell_k$. Precisely, the function first maps σ to $s = \hat{\mathsf{H}}(\mathsf{apk}, \sigma)$ where $s \in \{0,1\}^\lambda$, then maps s to $K = \mathsf{H}(s, c)$, and then truncates K to h bits. We can now precompute tables $\mathsf{Tab}_1, \ldots, \mathsf{Tab}_T$, where each table Tab_t is for fresh ciphertext $c = c_t \xleftarrow{\$} \{0,1\}^{\ell_c}$. In Figure 6, we describe algorithms $\mathsf{BuildTable}$ and $\mathsf{TryInvert}$ for implementing this approach. Namely, $\mathsf{BuildTable}$ takes as input a ciphertext c and creates a table. It will be called T times in the preprocessing step of our attack. Algorithm $\mathsf{TryInvert}$ will be called in the online phase to invert the function for a given key K and a given table. It returns (potentially more than one) preimage which can then be tried as a potential (part of the) secret key.

Attack Description. We present our attack in Figure 7. As outlined above, during preprocessing (algorithm ASetup), the attacker samples T distinct ciphertexts $c_1, \ldots, c_t$ and computes tables $\mathsf{Tab}_1, \ldots, \mathsf{Tab}_T$. Key generation is subverted as in Section 4.2, i.e., $s = \hat{\mathsf{H}}(\mathsf{apk}, \sigma)$, and decapsulation is not subverted. Then, in the online phase (algorithm $\mathsf{AOnline}$), the attacker iterates over all T tables and tries to find the secret key using each table Tab_t as follows: first it obtains K_t by submitting c_t to the decapsulation oracle. The attacker will assume that K_t is an implicit rejection key, i.e., it has the form $K_t = \mathsf{H}(s, c_t) = \mathsf{H}(\hat{\mathsf{H}}(\mathsf{apk}, \sigma), c_t)$. Then, the attacker applies algorithm $\mathsf{TryInvert}$ on table Tab_t. The algorithm returns a (potentially empty) set of preimages of the first h bits of K_t. Note, however, that this set contains false positives, in a sense that (1) not all preimages of the first h bits are also preimages

[1]The assumption $h \leq \ell_k$ is indeed natural: we have $h \leq \ell_s$ by definition, and $\ell_s \leq \ell_k$ holds naturally if we assume that sk' is the seed for a pseudorandom number generator that generates the actual key pair.

```
Alg BuildTable(apk, c)
for i ∈ [H] :
    x_{i,0} <-$ {0,1}^h
    for j ∈ [W] :  x_{i,j} := F(apk, c, x_{i,j-1})
    sp_i := x_{i,0},  ep_i := x_{i,W}
return Tab := (sp_i, ep_i)_{i=1}^H

Alg F(apk, c, x ∈ {0,1}^h)
s := Ĥ(apk, x),  K := H(s, c)
parse (y, y') := K,  y ∈ {0,1}^h
return y

Alg TryInvert(apk, c, Tab, K)
parse (sp_i, ep_i)_{i=1}^H := Tab
parse (y_1, y') := K,  y_1 ∈ {0,1}^h
Pre := ∅
for j ∈ [W] :
    I := {i ∈ [H] | y_j = ep_i}
    Pre := Pre ∪ {F^{W-j}(sp_i) | i ∈ I}
    y_{j+1} := F(apk, c, y_j)
return Pre
```

Figure 6: Algorithms BuildTable, TryInvert used in our attack in Section 4.3. The algorithms implement a time-memory trade-off for inverting the function mapping $\sigma \in \{0,1\}^h$ to the first h bits of $\mathsf{H}(\hat{\mathsf{H}}(\mathsf{apk}, \sigma), c)$. Here, $\hat{\mathsf{H}}\colon \{0,1\}^* \to \{0,1\}^\lambda$ is a random oracle and $H, W \in \mathbb{N}$ are parameters.

of the full key K_t, and (2) due to collisions in H or $\hat{\mathsf{H}}$, there could be valid preimages of K_t which are not related to sk' at all. The attacker can easily rule out false positives of category (1) (see set Pre' in algorithm AOnline). To rule out false positives of category (2), the attacker has to try to find a valid sk' by doing an exhaustive search over $2^{\ell_s - h}$ values.

```
Alg ASetup
apk <-$ {0,1}^λ
for t ∈ [T] :
    c_t <-$ {0,1}^{ℓ_c} \ {c_1, ..., c_{t-1}}
    Tab_t <- BuildTable(apk, c_t)
ask := (apk, (c_t, Tab_t)_{t=1}^T)
return (apk, ask)

Alg ADecaps(apk, sk', s, c)
return K := H(s, c)

Alg AGen(apk, sk')
parse (σ, σ') := sk',  σ ∈ {0,1}^h
return s := Ĥ(apk, σ)

Alg FinalSearch(pk, σ)
for σ' ∈ {0,1}^{ℓ_s - h} :
    sk' := (σ, σ') ∈ {0,1}^{ℓ_s}
    if (pk, sk') ∈ Gen :  return sk'
return ⊥

Alg AOnline^{Dec,H,G,Ĥ}(pk, ask)
parse (apk, (c_t, Tab_t)_{t=1}^T) := ask
for t ∈ [T] :
    K_t := Dec(c_t)
    Pre := TryInvert(apk, c_t, Tab_t, K_t)
    Pre' := {σ ∈ Pre | H(Ĥ(apk, σ), c_t) = K_t}
    for σ ∈ Pre' :
        sk' := FinalSearch(pk, σ)
        if sk' ≠ ⊥ :  return sk'
return ⊥
```

Figure 7: Attack scheme AS = (ASetup, AGen, ADecaps, AOnline) and helper algorithms PrefixSearch, FinalSearch subverting only key generation using preprocessing. Here, $\hat{\mathsf{H}}\colon \{0,1\}^* \to \{0,1\}^\lambda$ is a random oracle. The attack has parameters $h \le \min\{\ell_s, \lambda, \ell_k\}$ and $T \in \mathbb{N}$ that influence its efficiency and success probability.

Attack Analysis. Compared to our attack in Section 4.2, only the preprocessing and online phase of the attack have changed and subversion remained the same. Therefore, the attack is undetectable with the same arguments as in Section 4.2, but giving an upper bound on the running time and a lower bound on the success probability requires more care.

Lemma 7 (Offline Complexity)**.** *Consider the attack scheme* AS *in Figure 7. Then, algorithm* ASetup *computes at most $2HWT$ hash evaluations. Further, the size of the secret key is $\lambda + T(\ell_c + 2Hh)$ bits.*

Proof. This easily follows because ASetup invokes BuildTable T times, and one invocation of BuildTable evaluates the function F HW times, where one invocation of F consists of two hashes. Further, recall that the secret key consists of $\mathsf{apk} \in \{0,1\}^\lambda$ and T tables, which each are represented by a ciphertext and H starting points and endpoints. □

Lemma 8 (Online Complexity)**.** *Consider the attack scheme* AS *in Figure 7 and assume* $h \leq \min\{\ell_s, \lambda\}$ *and* $h < \ell_k$*. Then, algorithm* $\mathsf{AOnline}$ *issues at most* T *decapsulation queries. Further, in expectation, it computes at most*

$$2TW + 2^{-h-1}TH\left(W(W-1) + \frac{1}{3}(W-1)W(W+1)\right)$$

hash evaluations and checks $(\mathsf{pk}, \mathsf{sk}') \in \mathsf{Gen}$ *at most* $2\ell_k/(\ell_k - h) \cdot 2^{\ell_s - h}$ *times, where the expectation is taken over the randomness of* AS*, the random oracles, and the the game.*

Proof. It is clear that $\mathsf{AOnline}$ issues at most T decapsulation queries, namely, one for each table. Analyzing the expected number of hashes and key checks needs more care. By linearity of expectation, we can focus on a fixed iteration $t \in [T]$ of the main loop in algorithm $\mathsf{AOnline}$, i.e., only focus on table Tab_t, and multiply the result by T. So, fix $t \in [T]$ arbitrarily. We first count the expected number of hashes in iteration t. Recall that this iteration consists of (1) calling the decapsulation oracle, (2) running algorithm $\mathsf{TryInvert}$, (3) filtering the output Pre of $\mathsf{TryInvert}$ to get Pre', and (4) invoking $\mathsf{FinalSearch}$ for every entry in Pre'. Step (1) and (4) do not require any hash evaluations. Denote random variables modeling the number of hash evaluations caused by (2) and (3) by $\mathsf{HE}_2, \mathsf{HE}_3$, respectively. To bound the expectation of HE_2, consider running algorithm $\mathsf{TryInvert}$ and denote by Pre_j for $j \in [W]$ the set that is added to Pre during the jth iteration of $\mathsf{TryInvert}$'s main loop. Observe that

$$\mathbb{E}\left[\mathsf{HE}_2\right] \leq \sum_{j=1}^{W} \left(|\mathsf{Pre}_j| \cdot (W-j) \cdot 2 + 2\right) = 2W + 2\sum_{j=1}^{W} |\mathsf{Pre}_j| \cdot (W-j),$$

where we used that one evaluation of F costs two hashes. To bound the expectation of HE_3, note that filtering costs two hashes per entry in Pre, i.e.,

$$\mathbb{E}\left[\mathsf{HE}_3\right] \leq 2|\mathsf{Pre}| \leq 2\sum_{j=1}^{W} |\mathsf{Pre}_j|.$$

In combination, we get

$$\mathbb{E}\left[\mathsf{HE}_2 + \mathsf{HE}_3\right] \leq 2W + 2\sum_{j=1}^{W} \mathbb{E}\left[|\mathsf{Pre}_j|\right] \cdot (W - j + 1).$$

Next, we fix $j \in [W]$ and want to bound the expectation of $|\mathsf{Pre}_j|$. For that, let $y_1, \ldots, y_j$ be as in algorithm $\mathsf{TryInvert}$, i.e., y_1 denotes the first h bits of K_t, $y_2 = \mathsf{F}(\mathsf{apk}, c_t, y_1)$, and so on. For ease of notation, we now omit the first two inputs from F, meaning that we have $y_j = \mathsf{F}^{j-1}(y_1)$ and $\mathsf{ep}_i = \mathsf{F}^W(\mathsf{sp}_i)$. We have $\mathbb{E}\left[|\mathsf{Pre}_j|\right] \leq \sum_{i=1}^{H} \Pr\left[y_j = \mathsf{ep}_i\right]$ and for every $i \in [H]$, the probability of event $y_j = \mathsf{ep}_i$ is

$$\begin{aligned}
& \Pr\left[y_j = \mathsf{ep}_i\right] \\
= \quad & \Pr\left[\mathsf{F}^{j-1}(y_1) = \mathsf{F}^W(\mathsf{sp}_i)\right] \\
\leq \quad & \Pr\left[y_1 = \mathsf{F}^{W-j+1}(\mathsf{sp}_i)\right] \\
& + \sum_{k=0}^{j-2} \Pr\left[\mathsf{F}^{j-1-k}(y_1) = \mathsf{F}^{W-k}(\mathsf{sp}_i) \wedge \mathsf{F}^{j-1-k-1}(y_1) \neq \mathsf{F}^{W-k-1}(\mathsf{sp}_i)\right] \\
\leq \quad & j2^{-h},
\end{aligned}$$

where we have used the independence of y_1 and sp_i. In combination, we get

$$\mathbb{E}\left[\mathsf{HE}_2 + \mathsf{HE}_3\right] \leq 2W + 2\sum_{j=1}^{W} jH2^{-h} \cdot (W - j + 1)$$

$$= 2\left(W + 2^{-h}H\sum_{j=1}^{W} j \cdot (W - j + 1)\right)$$

$$= 2\left(W + 2^{-h}H\frac{W(W-1)}{2} + 2^{-h}H\sum_{j=1}^{W} j \cdot (W - j)\right)$$

$$= 2\left(W + 2^{-h}H\frac{W(W-1)}{2} + 2^{-h}H\frac{(W-1)W(W+1)}{6}\right),$$

where we have used $\sum_{j=1}^{W} j \cdot (W-j) = ((W-1)W(W+1))/6$. Next, we bound the number of key checks, i.e., the number of times the algorithm checks $(\mathsf{pk}, \mathsf{sk}') \in \mathsf{Gen}$. Observe that the algorithm does at most $2^{\ell_s - h}$ such checks per entry in Pre'. Therefore, we need to bound the expected size of Pre'. To do so, we set $C := (\ell_k + h)/(\ell_k - h)$ and define the following event over the probability space induced by the random oracles.

- Event Large: This event occurs, if there exists a $K \in \{0,1\}^{\ell_k}$ such that $|\Gamma_K| \geq C$, where $\Gamma_K := \{\sigma \in \{0,1\}^h \mid \mathsf{H}(\hat{\mathsf{H}}(\mathsf{apk}, \sigma), c_t) = K\}$.

We can bound the probability of Large as follows:

$$\Pr\left[\mathsf{Large}\right] \leq \sum_{K \in \{0,1\}^{\ell_k}} \Pr\left[|\Gamma_K| \geq C\right]$$

$$\leq \sum_{K \in \{0,1\}^{\ell_k}} \binom{2^h}{C} 2^{-\ell_k \cdot C} \leq 2^{\ell_k} \cdot 2^{h \cdot C} \cdot 2^{-\ell_k \cdot C} = 2^{\ell_k + C(h - \ell_k)}.$$

With that, we can bound the expected size of Pre':

$$\mathbb{E}\left[|\mathsf{Pre}'|\right] = \mathbb{E}\left[|\mathsf{Pre}'| \mid \mathsf{Large}\right] \cdot \Pr\left[\mathsf{Large}\right] + \mathbb{E}\left[|\mathsf{Pre}'| \mid \neg\mathsf{Large}\right] \cdot \Pr\left[\neg\mathsf{Large}\right]$$
$$\leq 2^h \cdot 2^{\ell_k + C(h-\ell_k)} + C.$$

Using the definition of C, the term above is at most $C + 1 = 2\ell_k/(\ell_k - h)$, finishing the proof. □

Lemma 9 (Advantage)**.** *Assume that* PKE *has spreadness* $\gamma \in [0,1]$ *and consider the attack scheme* AS *in Figure 7. Then, we have*

$$\mathsf{Adv}^{\mathsf{SUB\text{-}KR\text{-}CCA}}_{\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} \geq (1-\gamma)\left(1 - 2^{-h} \cdot \sum_{i=1}^{H}\sum_{j=1}^{W}\left(1 - \frac{iW}{2^h}\right)^j\right)^T$$

$$\geq (1-\gamma)\left(1 - 2^{-h} \cdot HW\left(1 - \frac{HW}{2^h}\right)^W\right)^T.$$

Proof. Recall that the probability space we consider is defined by the random oracles, the key apk, the ciphertexts $c_1, \ldots, c_T$, and the key pair $(\mathsf{pk}, \mathsf{sk}')$ for which the attack aims to find sk'. For our analysis, we denote the first h bits of sk' by σ^* and define the following:

- Event Win: This event occurs, if the attack finds sk'.
- Set Cover_t for each $t \in [T]$: This is the set of elements covered by table $t \in [T]$, namely,

$$\mathsf{Cover} := \left\{x_{i,j}^{(t)} \;\middle|\; i \in [H] \wedge j + 1 \in [W]\right\},$$

  where $x_{i,j}^{(t)}$ denotes the element in the ith row and jth column of table t.

- Index $t^* \in [T] \cup \{\bot\}$. This index is $t^* = \bot$ if there is no $t \in [T]$ with $\sigma^* \in \mathsf{Cover}_t$. Otherwise, it is the minimum such t.
- Event $\mathsf{GoodTab}_t$ for $t \in [T]$: This event occurs if the ciphertext c_t is invalid, i.e., $\mathsf{Decaps}(\mathsf{sk}', c_t) = \bot$ or $c_t \neq \mathsf{Enc}(\mathsf{pk}, \mathsf{Decaps}(\mathsf{sk}', c_t); \mathsf{G}(\mathsf{Decaps}(\mathsf{sk}', c_t)))$.

The goal of our analysis is to give a lower bound on the probability of Win. This is done as follows: we first observe that if σ^* is in one of the tables and not an endpoint, and this table is associated with a ciphertext that triggers implicit rejection, then the attack will find σ^* and will also find sk'. Namely,

$$\Pr[\mathsf{Win}] \geq \Pr[t^* \neq \bot \wedge \mathsf{GoodTab}_{t^*}] = \Pr[\mathsf{GoodTab}_{t^*} \mid t^* \neq \bot] \cdot \Pr[t^* \neq \bot].$$

We will analyze these two terms separately. For analyzing the former term, we will assume that the randomness is taken only over the keys and G and everything else is fixed, including $\hat{\mathsf{H}}$ and H. For analyzing the latter term, we will assume that the randomness is taken only over random oracles $\hat{\mathsf{H}}$ and H and everything else, including σ^*, the keys, and G is fixed. It can easily be seen[2] that

$$\Pr_{\mathsf{pk},\mathsf{sk}',\mathsf{G}}[\mathsf{GoodTab}_{t^*} \mid t^* \neq \bot] \geq 1 - \gamma.$$

Hence, we can focus on upper bounding the probability of $t^* = \bot$. We have

$$\Pr_{\mathsf{H},\mathsf{G}}[t^* = \bot] = \Pr[\forall t \in [T] : \sigma^* \notin \mathsf{Cover}_t] = \prod_{t \in [T]} \Pr[\sigma^* \notin \mathsf{Cover}_t].$$

For every $t \in [T]$, we can bound the respective term via

$$\Pr[\sigma^* \notin \mathsf{Cover}_t] \leq \left(1 - 2^{-h} \cdot \mathbb{E}[|\mathsf{Cover}_t|]\right).$$

Now, we have reduced the proof to lower bounding the expected size[3] of Cover_t for a fixed $t \in [T]$. The reader shall keep in mind that the randomness space is defined only by the random oracles and everything else is treated as fixed. We first define another class of events:

- Event $\mathsf{New}_{i,j}$ for $i \in [H]$ and $j\{0, \ldots, W-1\}$: This event occurs, if the element $x_{i,j}^{(t)}$ does not occur in a previous row or a previous column in row i. More formally, it occurs if there is no pair $(i', j') \in [H] \times \{0, \ldots, W-1\}$ with $i' < i$ and $x_{i',j'}^{(t)} = x_{i,j}^{(t)}$ or with $j' \leq j$ and $x_{i,j'}^{(t)} = x_{i,j}^{(t)}$.

It is clear that the probability of $\mathsf{New}_{i,j}$ is at least the probability that all $\mathsf{New}_{i,j'}$ for $j' \leq j$ hold. For every such event $\mathsf{New}_{i,j'}$, the probability that it holds conditioned on that all previous ones hold is at least $1 - iW2^{-h}$, as there are at most iW many of the 2^h elements that are already covered. This means that

$$\Pr[\mathsf{New}_{i,j}] \geq \left(1 - \frac{iW}{2^h}\right)^{j+1}.$$

By linearity of expectation, we get

$$\mathbb{E}[|\mathsf{Cover}_t|] = \sum_{i=1}^{H} \sum_{j=0}^{W-1} \Pr[\mathsf{New}_{i,j}] \geq \sum_{i=1}^{H} \sum_{j=1}^{W} \left(1 - \frac{iW}{2^h}\right)^{j}.$$

In combination, we get the desired bound. □

Lemma 10 (Undetectability)**.** *Consider the attack scheme* AS *in Figure 7. Then, for any algorithm* $\mathcal{D}$ *that makes at most* Q *queries to random oracle* $\hat{\mathsf{H}}$*, we have*

$$\mathsf{Adv}^{\mathsf{dist\text{-}kp}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} \leq \frac{Q}{2^\lambda}, \quad \mathsf{Adv}^{\mathsf{dist\text{-}o}}_{\mathcal{D},\mathsf{AS},\mathsf{PKE},\mathsf{H},\mathsf{G}} = 0.$$

Proof. Note that the way subversion works is identical to what we have done in Section 4.2. Therefore, the proof is identical to the proof of Lemma 6. □

[2] See the proof of Lemma 2 for a detailed proof similar statement.
[3] This part of the analysis is similar to the analysis in [Hel80].

Concrete Example: Kyber. Setting $W = H = T = 2^{h/3}$, we can approximate the advantage of the attack according to Lemma 9 by $(1-\gamma) \cdot (1/e)^{1/e}$, which is approximately $0.7 \cdot (1-\gamma)$. It remains to find a good choice for h to minimize the attack complexity. For that, our goal is that the final brute-force search $\mathsf{FinalSearch}$ takes approximately as long as the rest of the attack. This leads to $\ell_s - h = 2h/3$, i.e., $h = 3\ell_s/5$. With $\ell_s = 256$ for Kyber, we get the numbers we have in Table 1.

On Quantum Speed-Ups. Using a quantum computer, we can speed up the two exhaustive searches in our attack in Section 4.2 using Grover's algorithm [Gro96], which results in taking a square root of the complexity. Interestingly, this is more efficient than a similar quantum speed up for the time-memory trade-off in this section. The reason is that known quantum equivalents to Hellman tables, e.g., [DKRS21], are only more efficient than Grover if we are interested in inverting one out of many images of a function. We currently do not see how to leverage this for our attack.

References

[ABD+21] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber algorithm specifications and supporting documentation, version 3.02. 2021. (Cited on page 3, 4, 5, 10.)

[ABR98] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. DHIES: An encryption scheme based on the Diffie-Hellman problem. Contributions to IEEE P1363a, September 1998. (Cited on page 13.)

[AMV15] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 364–375. ACM Press, October 2015. (Cited on page 5.)

[BBN+09] Mihir Bellare, Zvika Brakerski, Moni Naor, Thomas Ristenpart, Gil Segev, Hovav Shacham, and Scott Yilek. Hedged public-key encryption: How to protect against bad randomness. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 232–249. Springer, Heidelberg, December 2009. (Cited on page 5.)

[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. (Cited on page 3, 5.)

[BFS16] Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg, December 2016. (Cited on page 5.)

[BH15] Mihir Bellare and Viet Tung Hoang. Resisting randomness subversion: Fast deterministic and hedged public-key encryption in the standard model. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 627–656. Springer, Heidelberg, April 2015. (Cited on page 5.)

[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. (Cited on page 3, 5.)

[BJK15] Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1431–1440. ACM Press, October 2015. (Cited on page 5.)

[BPR14] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 1–19. Springer, Heidelberg, August 2014. (Cited on page 5.)

[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. (Cited on page 3, 4, 5, 7.)

[DFMS22] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Online-extractability in the quantum random-oracle model. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 677–706. Springer, Heidelberg, May / June 2022. (Cited on page 3.)

[DFP15] Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass surveillance. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 579–598. Springer, Heidelberg, March 2015. (Cited on page 5.)

[DHK+21] Julien Duman, Kathrin Hövelmanns, Eike Kiltz, Vadim Lyubashevsky, and Gregor Seiler. Faster lattice-based KEMs via a generic fujisaki-okamoto transform using prefix hashing. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2722–2737. ACM Press, November 2021. (Cited on page 5.)

[DKRS21] Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. Quantum time/memory/data tradeoff attacks. Cryptology ePrint Archive, Report 2021/1561, 2021. `https://eprint.iacr.org/2021/1561`. (Cited on page 20.)

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984. (Cited on page 6, 13.)

[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. (Cited on page 3, 4, 5, 6, 7.)

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 26(1):80–101, January 2013. (Cited on page 4.)

[Fuc18] Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347. Springer, Heidelberg, March 2018. (Cited on page 5.)

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. (Cited on page 3.)

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *28th ACM STOC*, pages 212–219. ACM Press, May 1996. (Cited on page 4, 20.)

[Hel80] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory*, 26(4):401–406, 1980. (Cited on page 4, 15, 19.)

[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. (Cited on page 3, 5, 6, 7.)

[HHM22] Kathrin Hövelmanns, Andreas Hülsing, and Christian Majenz. Failing gracefully: Decryption failures and the fujisaki-okamoto transform. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 414–443. Springer, Heidelberg, December 2022. (Cited on page 3, 5.)

[HKSU20] Kathrin Hövelmanns, Eike Kiltz, Sven Schäge, and Dominique Unruh. Generic authenticated key exchange in the quantum random oracle model. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 389–422. Springer, Heidelberg, May 2020. (Cited on page 3, 5.)

[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. (Cited on page 3, 5.)

[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. (Cited on page 3, 5.)

[NIS17] NIST. Post-Quantum Cryptography. https://csrc.nist.gov/projects/post-quantum-cryptography, 2017. Accessed: 2024-02-07. (Cited on page 3.)

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990. (Cited on page 3.)

[Per12] Edoardo Persichetti. *Improving the efficiency of code-based cryptography.* PhD thesis, ResearchSpace Auckland, 2012. (Cited on page 4.)

[RBC+22] Prasanna Ravi, Shivam Bhasin, Anupam Chattopadhyay, Aikata, and Sujoy Sinha Roy. Backdooring post-quantum cryptography: Kleptographic attacks on lattice-based KEMs. Cryptology ePrint Archive, Report 2022/1681, 2022. https://eprint.iacr.org/2022/1681. (Cited on page 5.)

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. (Cited on page 6.)

[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. (Cited on page 3.)

[RTYZ16] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the power of kleptographic attacks. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 34–64. Springer, Heidelberg, December 2016. (Cited on page 5.)

[RTYZ17] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Generic semantic security against a kleptographic adversary. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 907–922. ACM Press, October / November 2017. (Cited on page 5.)

[SAB+22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. (Cited on page 3.)

[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. (Cited on page 3, 5.)

[TY17] Qiang Tang and Moti Yung. Cliptography: Post-snowden cryptography. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2615–2616. ACM Press, October / November 2017. (Cited on page 5.)

[YY96] Adam Young and Moti Yung. The dark side of "black-box" cryptography, or: Should we trust capstone? In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 89–103. Springer, Heidelberg, August 1996. (Cited on page 3, 5.)

[YY97a] Adam Young and Moti Yung. Kleptography: Using cryptography against cryptography. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 62–74. Springer, Heidelberg, May 1997. (Cited on page 3, 5.)

[YY97b] Adam Young and Moti Yung. The prevalence of kleptographic attacks on discrete-log based cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 264–276. Springer, Heidelberg, August 1997. (Cited on page 5.)

[YY01] Adam Young and Moti Yung. Bandwidth-optimal kleptographic attacks. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 235–250. Springer, Heidelberg, May 2001. (Cited on page 5.)

Appendix

A Script for Attack Complexity and Advantage

Listing 1: Python script to estimate the complexity and advantage of our three attacks for the example of Kyber, as presented in Table 1.

```python
#!/usr/bin/env python

import math
import sys
from tabulate import tabulate

#-------------------------------------------------------------------------#
#                          KYBER PARAMETERS                               #
#-------------------------------------------------------------------------#

# lsk: length of secret key, or precisely, the seed to generating it
lsk = 32*8

# lseed: length of the seed for implicit rejection
lseed = 32*8

# lk: length of KEM session keys
lk = 32*8

# lc: length of a kyber ciphertext, precisely, Kyber1024
lc = 1568*8

# spread: upper bound on spreadness
spread = 0.001

#-------------------------------------------------------------------------#
#                               ATTACKS                                   #
#-------------------------------------------------------------------------#

# an attack is given by
#
# name:          name of the attack
# advantage:     lambda for computing the advantage of the attack
# memory:        lambda for computing the memory complexity of the attack
# time_offline:  lambda for computing the offline complexity of the attack
# time_online:   lambda for computing the online complexity of the attack
#                for running times, we assume all basic operations (hash,
#                checking key pairs, etc) take one step
#
# all of these lambdas should take as input a kyber configuration


# we use hashed ElGamal encryption
# in this case, a ciphertext has size < 3 * 256
T = math.ceil(3.0 * 256.0 / lk)
decapsattack = {
        "name": "decaps",
        "memory": 256,
        "time_offline": 1,
        "time_online": T,
        "advantage": 1- T * spread,
}

h = lsk/2
secpar = 128
keygenattack = {
        "name": "key gen",
        "memory": secpar,
        "time_offline": 1,
        "time_online": 1 + 2**(h+1) + 2**(lsk-h),
        "advantage": 1.0 - spread - 2**(-secpar) - 2**(-lk),
}


h = int(lsk*3/5.0)
logW = h/3
logH = h/3
logT = h/3
# need that for approximation 1/e
assert(2*logW + logH == h)
assert(logW + logH + logT == h)

W = int(2**logW)
H = int(2**logH)
T = int(2**logT)


secpar = 128
preproattack = {
        "name": "preprocess",
        "memory": secpar + T*(lc + 2*H*h),
        "time_offline": 2*H*W*T,
        "time_online": 2*T*W + 2**(-h-1)*T*H*(W*(W-1)+((W-1)*W*(W+1))/3) + (2**(lsk - h)*2*lk / (lk-h)),
        "advantage": (1.0-spread)*((1.0/math.e)**(1/math.e))
}
attacks = [decapsattack, keygenattack, preproattack]


#-------------------------------------------------------------------------#
#                              TABLE GEN                                  #
#-------------------------------------------------------------------------#


# one row per attack
# each row contains: name, memory, time offline, time online, advantage
table = [["Attack", "Memory", "Time (Offline)", "Time (Online)", "Advantage"]]
```

```
for a in attacks:
        name = a["name"]
        mem = math.ceil(math.log2(a["memory"]))
        toff = math.ceil(math.log2(a["time_offline"]))
        ton =  math.ceil(math.log2(a["time_online"]))
        adv = a["advantage"]
        row = [name, mem, toff, ton, adv]
        table.append(row)


#-------------------------------------------------------------------------#
#                                USER OUTPUT                              #
#-------------------------------------------------------------------------#
print("")
print("Hint: to print the table in LaTeX code, add the option -l.")
print("Note: except advantage, all numbers are logarithms to base 2.")
print("")

opts = [opt for opt in sys.argv[1:] if opt.startswith("-")]

if "-l" in opts:
        print(tabulate(table, headers='firstrow', tablefmt='latex_raw', disable_numparse = True))
else:
        print(tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
```