Hybrid Obfuscated Key Exchange and KEMs

Felix Günther
IBM Research Europe – Zurich
mail@felixguenther.info

Michael Rosenberg
Cloudflare
research@mrosenberg.pub

Douglas Stebila
University of Waterloo
dstebila@uwaterloo.ca

Shannon Veitch
ETH Zurich
shannon.veitch@inf.ethz.ch

Version 1.0, March 3, 2025

Abstract

Hiding the metadata in Internet protocols serves to protect user privacy, dissuade traffic analysis, and prevent network ossification. *Fully encrypted protocols* require even the initial key exchange to be obfuscated: a passive observer should be unable to distinguish a protocol execution from an exchange of random bitstrings. Deployed obfuscated key exchanges such as Tor's pluggable transport protocol `obfs4` are Diffie–Hellman-based, and rely on the Elligator encoding for obfuscation. Recently, Günther, Stebila, and Veitch (CCS '24) proposed a post-quantum variant `pq-obfs`, using a novel building block called *obfuscated key encapsulation mechanisms* (OKEMs): KEMs whose public keys and ciphertexts look like random bitstrings.

For transitioning real-world protocols, pure post-quantum security is not enough. Many are taking a *hybrid* approach, combining traditional and post-quantum schemes to hedge against security failures in either component. While hybrid KEMs are already widely deployed (e.g., in TLS 1.3), existing hybridization techniques fail to provide hybrid obfuscation guarantees for OKEMs. Further, even if a hybrid OKEM existed, the `pq-obfs` protocol would still not achieve hybrid obfuscation.

In this work, we address these challenges by presenting the first OKEM combiner that achieves hybrid IND-CCA security with hybrid ciphertext obfuscation guarantees, and using this to build `Drivel`, a modification of `pq-obfs` that is compatible with hybrid OKEMs. Our OKEM combiner allows for a variety of practical instantiations, e.g., combining obfuscated versions of DHKEM and ML-KEM. We additionally provide techniques to achieve unconditional public key obfuscation for LWE-based OKEMs, and explore broader applications of hybrid OKEMs, including a construction of the first hybrid password-authenticated key exchange (PAKE) protocol secure against adaptive corruptions in the UC model.

Keywords: KEM combiners, hybrid, obfuscation, anonymity, key exchange, ML-KEM, quantum-safe

Contents

1 Introduction

Increasingly, Internet protocols are hiding not just message contents, but the protocol metadata itself, due to metadata being repeatedly leveraged to violate user security and privacy, and to degrade connectivity [Hal24, XAR$^+$24, WMSM11, And12]. As a pertinent example, network censors often block traffic based on certain identifiable (plaintext) features of the protocol [WSS$^+$23]. To evade this form of censorship, circumvention tools often encrypt metadata to avoid protocol fingerprinting and blocklists [The19, sha23, vme19]. These *fully encrypted protocols* [FJ24], such as the `obfs4` protocol integrated in the Tor ecosystem [The19], produce a stream of random-looking bytes intended to avoid classification. Such streams can also be used in steganography due to their lack of structure [WJJS23]. Another motivation for encrypting protocol metadata, e.g., in pseudorandom cTLS [SP22], is to prevent *network ossification*, whereby network middleboxes expect a specific protocol mode (such as TLS 1.2) and fail to parse and route traffic of newer protocol versions (such as TLS 1.3 without its intentional 1.2 compatibility). If protocol metadata appears to a middlebox as a random string of bytes, then no parsers can be written in the first place. Random-looking metadata as an anti-ossification tool is already widely deployed, appearing in the QUIC protocol [IT21], where elements of the initial packet are encrypted to make the wire image of the protocol closer to pseudorandom, and in TLS Encrypted Client Hello [ROSW24], where the server "accept" response is encoded as a truncated MAC and hidden in the randomness sent from server to client. Both these techniques are also employed to improve privacy guarantees.

Recent formalizations capture the guarantees provided by fully encrypted protocols in the context of censorship circumvention [FJ24, GSV24]. A shared secret key allows encrypting (meta)data into random-looking strings of bytes. However, it is less obvious how to make the initial key exchange of a connection look random—this is what *obfuscated key exchange* protocols aim to achieve.

1.1 Post-quantum Obfuscated Key Exchange and Beyond

As timelines for cryptographically relevant quantum computers shorten, there is an increased need to update current obfuscated key exchange protocols which derive their security from elliptic curve Diffie–Hellman. Recent work [GSV24] proposes a quantum-safe variant of `obfs4` called `pq-obfs`. The construction relies on the newly defined *obfuscated key encapsulation mechanism* (OKEM): a KEM that supports random-looking encoding of public keys and ciphertexts. That work shows how to construct an efficient OKEM from the Module Learning with Errors (MLWE) assumption by introducing the Kemeleon encoding which—akin to the Elligator2 encoding for elliptic curve Diffie–Hellman [BHKL13]—turns ML-KEM public keys and ciphertexts into random strings.

Due to caution over fully switching to newer assumptions, some governments [Ger24, Fre22] and cloud providers [Wes24] opt instead for *hybrid* constructions, i.e., constructions whose assumptions combine the hardness of two input primitives, usually one post-quantum and one well-established. The reasons for this approach are twofold. First, using two hardness assumptions hedges risk of mathematical advances, since the resulting construction is secure even if one of the underlying assumptions fails, as happened with NIST PQC Round 3 alternates and a finalist [BBC$^+$22, Beu22, CD23]. Second, it hedges risk of implementation errors, since newer post-quantum implementations may be more prone to bugs [GJN20, GHJ$^+$22, HSC$^+$23, BBB$^+$24]. For these reasons, hybrid protocols, such as the

X25519MLKEM768 key agreement protocol for TLS 1.3 have already been deployed to millions worldwide [Rad]. It would thus be ideal to migrate obfuscated key exchange protocols to post-quantum using a hybrid solution rather than a purely lattice-based one.

To date, the work on building hybrid KEMs and key exchange has focused primarily on IND-CCA security [GHP18, WW24, BCD$^+$24, BBF$^+$19]. And although it has already been argued that a broader study of security properties should be considered for post-quantum KEMs [GMP22], works exploring additional properties in the hybrid setting are very sparse [PG25]. To this end, we initiate a study of obfuscation and anonymity guarantees in the hybrid setting, towards constructing quantum-safe metadata-hiding protocols that are in line with modern techniques of hybridization.

1.2 Barriers to Hybrid Obfuscated Key Exchange

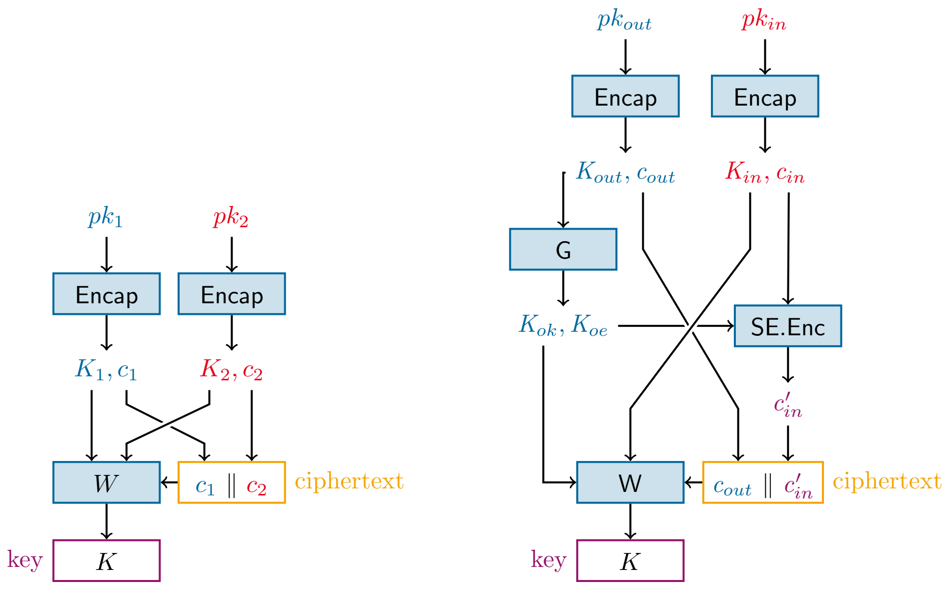
One might think that a straightforward approach to construct hybrid obfuscated key exchange is to instantiate pq-obfs with a hybrid OKEM combining two OKEMs using a standard KEM combiner. This however does not work, for two reasons.

Firstly, there are no OKEM-specific hybridization methods, and the existing methods for KEM hybridization do not produce hybrid OKEMs. A KEM combiner takes as input two KEMs and produces a single, hybrid KEM whose security is maintained as long as one of the two input KEMs remains secure. Existing combiner techniques [GHP18, WW24, BCD$^+$24] do not apply to the obfuscation properties of OKEMs, where the goal is to have public keys and ciphertexts indistinguishable from random strings. Take for example the Xyber hybrid KEM [WW24], which encapsulates by performing a Kyber encapsulation and X25519 key agreement in parallel, and where the final shared secret is a key derived from the two resulting shared secrets. While parallel combiners like this provide hybrid IND-CCA security, they do not provide hybrid obfuscation for OKEMs: if an adversary receives two concatenated public keys or ciphertexts, then they need only observe a non-uniformity in *one* of them to determine that the joint public key/ciphertext is not random. Most existing combiners are in this parallel style, depicted in Figure 1(a), which even when applied to OKEMs only provides obfuscation as strong as the *weakest* of the underlying obfuscation assumptions.

Secondly, even if there were an OKEM hybridization technique, there is still a hurdle in using it to make pq-obfs hybrid. pq-obfs requires its OKEM to have *public key uniformity*: public keys must appear uniformly random. Since a KEM public key must be available for encapsulation prior to any other communication, it appears the only way to represent the combined public key is as the concatenation of the underlying public keys, or something equivalent. As a consequence, constructing a combined OKEM with hybrid public key uniformity seems to require that the two underlying OKEMs have unconditional public key uniformity. Unconditional public key uniformity, however, is uncommon for post-quantum (O)KEMs, where public keys have structure that becomes distinguishable from random if the underlying hardness assumption breaks down. This bars the path to make pq-obfs into a hybrid obfuscated key exchange using standard components.

1.3 Contributions

We discuss how we overcome these limitations to achieve hybrid obfuscated key exchange, and introduce an additional application in hybrid password-authenticated key exchange (PAKE) protocols.



(a) Classic parallel-style KEM combiner (b) Our Outer-Encrypts-Inner Nested Combiner OEINC

Figure 1: Structure of Encap in KEM Combiners. W is a split-key PRF, G is a pseudorandom generator, and SE is symmetric encryption.

OKEM combiner. Our first contribution is OEINC ("oink"), our *Outer-Encrypts-Inner Nested Combiner* for OKEMs which yields hybrid IND-CCA security and ciphertext obfuscation (without public key obfuscation) properties. The combiner is not fully generic, as it requires one of the component OKEMs to have statistically uniform ciphertexts. This allows us to use a nested approach in which we leverage the stronger OKEM as the *outer* one, using its shared secret towards encrypting the ciphertext of the weaker, *inner* OKEM. Our nested approach is depicted in Figure 1(b). This provides hybrid guarantees for ciphertext uniformity and strong pseudorandomness (which in turn implies hybrid anonymity).

We show that the necessary statistical uniformity of ciphertexts can be achieved using the DHKEM construction from HPKE [BBLW22], over a prime-order group such as P-256 or Ristretto [NIS23, LHT16], with the Elligator2 encoding algorithm [BHKL13]. This obfuscated variant of DHKEM can then be combined with a post-quantum OKEM, such as ML-Kemeleon [GSV24], to achieve the desired hybrid guarantees. We use OEINC as a building block for the following applications.

Application: Hybrid obfuscated key exchange. Naturally, we would like to apply our OKEM combiner to obtain a hybrid obfuscated key exchange protocol, but, as noted above, hybrid `pq-obfs` would require hybrid public key uniformity, which OEINC does not achieve and generally seems elusive for hybrid OKEMs with post-quantum security.

We present `Drivel`, a modification to the `pq-obfs` protocol which permits the use of OKEMs *without* needing to obfuscate public keys. Our key insight is that public keys need not be sent in the clear, but can instead be encrypted with intermediate secrets in the key exchange. We describe the new protocol in detail and prove that `Drivel` achieves the same security guarantees as `pq-obfs`.

Application: Hybrid PAKE with adaptive corruptions. Finally, we consider how to apply our combiner to construct a hybrid password-authenticated key exchange (PAKE) protocol. Existing hybrid PAKEs are secure only in the static corruption model, where the adversary may not corrupt any parties during the protocol execution [HR24, LL24]. Prior work [HR24] asked whether there exists a method for constructing adaptively secure hybrid PAKEs. We answer this question in the affirmative.

We observe that CAKE [BCP$^+$23], which is proven secure in the Universal Composability model [Can01] with adaptive corruptions, can be instantiated with any OKEM with ciphertext and public key uniformity. As previously noted, public key uniformity guarantees for combiners seem elusive unless both underlying OKEMs have unconditional public key uniformity. We describe a method to achieve unconditional public key uniformity in LWE-based OKEMs. We observe that some standards-track KEMs, such as $\mathsf{FrodoKEM}$ [BCD$^+$16, NAB$^+$20], have a *dual*-LWE structure: both public keys and ciphertexts are LWE samples. This is done to optimize ciphertext size, but is not strictly necessary. We show how removing this optimization (reverting back to earlier ideas of [Reg05]) yields a statistically uniform public key, at the expense of a $15\times$ increase in ciphertext size.

Combining this LWE-based OKEM with a classical OKEM with unconditional ciphertext uniformity via OEINC, and then using it in CAKE, we achieve hybrid PAKE. This is the first known hybrid PAKE that is secure under adaptive corruptions.

2 Preliminaries

We write $y \leftarrow \mathsf{A}(z)$ to denote assigning y the output of a deterministic algorithm $\mathsf{A}(z)$. When A is non-deterministic, we write $y \leftarrow\!\!\$\, \mathsf{A}(z)$. Similarly, we write $y \leftarrow\!\!\$\, S$ to denote uniformly sampling an element from the set S.

A *pseudorandom function* (PRF) $\mathsf{F}\colon \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ treats the first input as a key and the second as a label. For a randomly chosen key, the outputs on an adversarially-chosen label of F should be indistinguishable from those of a random function on the label input. The *swap* of F is $\mathsf{F}'(y, x) = \mathsf{F}(x, y)$. We say that F is a *swap-PRF* if its swap F' is a PRF. A *dual-PRF* is both a PRF and a swap-PRF. We defer the standard definitions of PRF security, pseudorandom generator (PRG) security, and $\mathsf{OT\text{-}IND\$}$ security (one-time indistinguishability of ciphertexts from random) of symmetric encryption to Appendix A.

We adopt the definition of a *split-key pseudorandom function* from [GHP18], which is a generalization of a pseudorandom function to multiple key inputs that behaves like a random function if at least one of its key inputs is picked uniformly at random.

Definition 2.1 (Split-key pseudorandom function)**.** *A* split-key pseudorandom function $\mathsf{F}\colon \mathcal{K}_1 \times \cdots \times \mathcal{K}_n \times \mathcal{X} \to \mathcal{Y}$ *takes as input a finite number of keys in* $\mathcal{K}_1 \times \cdots \times \mathcal{K}_n$ *and a label in* $\mathcal{X}$ *and produces an output in* $\mathcal{Y}$*. We define the* split-key pseudorandom function (skPRF) *advantage of an adversary* $\mathcal{A}$ *against a function* F *as*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{F},i}(\mathcal{A}) := \; & \Pr\left[\mathcal{A}^{\mathsf{F}(\cdots,k_i,\cdots)}() \Rightarrow 1 \mid k_i \leftarrow\!\!\$\, \mathcal{K}_i\right] \\ & - \Pr\left[\mathcal{A}^{R(\cdots)}() \Rightarrow 1 \mid R \leftarrow\!\!\$\, \{\text{all functions} : \mathcal{K}_{n\setminus i} \times \mathcal{X} \to \mathcal{Y}\}\right],\end{aligned}$$

where $\mathcal{K}_{n\setminus i}$ *denotes* $\mathcal{K}_1 \times \cdots \times \mathcal{K}_{i-1} \times \mathcal{K}_{i+1} \times \cdots \times \mathcal{K}_n$*.*

2.1 Obfuscated KEMs

Günther, Stebila, and Veitch [GSV24] introduced notions of obfuscated key encapsulation mechanisms (OKEMs). We present equivalent definitions next, with slight modifications for simplicity.

Definition 2.2 (Key encapsulation mechanism). *A key encapsulation mechanism* $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap})$ *consists of three algorithms:*

- $\underline{\mathsf{KGen}}() \xrightarrow{\$} (sk, pk)$ *is a probabilistic* key generation *algorithm that generates a secret key sk and corresponding public key pk.*
- $\underline{\mathsf{Encap}}(pk) \xrightarrow{\$} (c, K)$ *is a probabilistic* encapsulation *algorithm that takes as input a KEM public key pk, and outputs a ciphertext c and shared secret K.*
- $\underline{\mathsf{Decap}}(sk, c) \to K$ *is a deterministic* decapsulation *algorithm that takes as input a secret key sk and ciphertext c, and outputs a shared secret K.*

Definition 2.3 (KEM encapsulation/decapsulation correctness). *We say that a KEM* $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap})$ *is* δ_{KEM}-correct *if*

$$\Pr\left[\mathsf{Decap}(sk, c) \neq K \;\middle|\; \begin{array}{c} (sk, pk) \leftarrow\!\$\; \mathsf{KGen}(), \\ (c, K) \leftarrow\!\$\; \mathsf{Encap}(pk) \end{array}\right] \leq \delta_{\mathsf{KEM}}.$$

IND-CPA, IND-CCA, SPR-CCA. We will consider KEMs with indistinguishability under chosen-plaintext or chosen-ciphertext attacks ($\mathsf{IND\text{-}CPA}$ / $\mathsf{IND\text{-}CCA}$), meaning that the shared secret is indistinguishable from random given a real ciphertext, as well as strong pseudorandomness under chosen-ciphertext attacks ($\mathsf{SPR\text{-}CCA}$) [Xag22], meaning that a real (ciphertext, shared secret) pair is indistinguishable from a random pair (in $\mathsf{SPR\text{-}CCA}$, pseudorandomness of the ciphertext is defined with respect to a simulator $\mathcal{S}$ defining a ciphertext target distribution). We provide the definitions of both these games in Figure 2 and define the respective advantages of an adversary $\mathcal{A}$ against the $X \in \{\mathsf{IND\text{-}CPA}, \mathsf{IND\text{-}CCA}, \mathsf{SPR\text{-}CCA}\}$ security of a KEM K as

$$\mathsf{Adv}^{X}_{\mathsf{K}}(\mathcal{A}) := 2 \cdot \Pr\left[\mathsf{G}^{X}_{\mathsf{K}}(\mathcal{A}) \Rightarrow 1\right] - 1.$$

Definition 2.4 (KEM public key collision probability). *Let* KEM *be a KEM. We define the* public key collision probability *of* KEM *for $n \in \mathbb{N}$ public keys as*

$$\mathsf{pkcoll}_{\mathsf{KEM}}(n) := \Pr\left[\begin{array}{c} pk_i = pk_j \\ \wedge\, i \neq j \end{array} \;\middle|\; \begin{array}{c} (sk_i, pk_i) \leftarrow\!\$\; \mathsf{KEM.KGen}() \\ \textit{for } i \in [1, n] \end{array}\right].$$

Definition 2.5 (Obfuscated KEM). *An* obfuscated key encapsulation mechanism (OKEM) $\mathsf{O} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{DecodePk})$ *with* obfuscated key length $\mathsf{ol} \in \mathbb{N}$ *and* obfuscated ciphertext length $\mathsf{cl} \in \mathbb{N}$ *consists of the following algorithms:*

- $\underline{\mathsf{KGen}}() \xrightarrow{\$} (sk, pk, \widehat{pk})$ *is a probabilistic key generation algorithm that generates a secret key sk, public key pk, and obfuscated public key $\widehat{pk} \in \{0,1\}^{\mathsf{ol}}$.*
- $\underline{\mathsf{Encap}}(pk) \xrightarrow{\$} (c, K)$ *is a probabilistic encapsulation algorithm that takes as input a KEM public key pk and outputs an (obfuscated) ciphertext $c \in \{0,1\}^{\mathsf{cl}}$ and key K.*

```
$\mathsf{G}^{\mathsf{IND\text{-}CCA}}_{\mathsf{K}}(\mathcal{A})$:
1  $b \leftarrow\$ \{0,1\}$
2  $(pk, sk) \leftarrow\$ \mathsf{KGen}()$
3  $(c^*, K_1^*) \leftarrow\$ \mathsf{Encap}(pk)$
4  $K_0^* \leftarrow\$ \mathcal{K}$
5  $b' \leftarrow\$ \mathcal{A}^{\mathcal{O}_{\mathsf{Decap}}(\cdot)}(pk, c^*, K_b^*)$
6  return $[\![b = b']\!]$

$\mathsf{G}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{K},\mathcal{S}}(\mathcal{A})$:
7  $b \leftarrow\$ \{0,1\}$
8  $(pk, sk) \leftarrow\$ \mathsf{KGen}()$
9  $(c_1^*, K_1^*) \leftarrow\$ \mathsf{Encap}(pk)$
10 $c_0^* \leftarrow\$ \mathcal{S}$; $K_0^* \leftarrow\$ \mathcal{K}$
11 $c^* \leftarrow c_b^*$
12 $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Decap}}(\cdot)}(pk, c_b^*, K_b^*)$
13 return $[\![b = b']\!]$

$\mathcal{O}_{\mathsf{Decap}}(c)$:
14 if $c = c^*$ then return $\bot$
15 $K \leftarrow \mathsf{Decap}(sk, c)$
16 return $K$
```

Figure 2: Security games for IND-CCA and SPR-CCA security of a KEM $\mathsf{K} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap})$ with key space $\mathcal{K}$. The IND-CPA game is obtained by removing the decapsulation oracle from IND-CCA.

- $\underline{\mathsf{Decap}}(sk, c) \to K$ *is a deterministic decapsulation algorithm that takes as input a secret key sk and (obfuscated) ciphertext c, and outputs a key K.*
- $\underline{\mathsf{DecodePk}}(\hat{pk}) \to pk$ *is a deterministic* decoding *algorithm that on input an obfuscated public key* $\hat{pk} \in \{0,1\}^{\mathsf{ol}}$ *outputs a public key pk.*

Note that the tuple of algorithms (KGen, Encap, Decap)*, when ignoring the obfuscated public key* $\widehat{pk}$ *output by* KGen*, is a KEM as in Definition 2.2.*

Beyond KEM correctness, as per Definition 2.3, we demand that public keys generated by KGen *can be successfully decoded:*

$$\Pr\left[\mathsf{DecodePk}(\widehat{pk}) = pk \;\middle|\; (sk, pk, \widehat{pk}) \leftarrow\$ \mathsf{KGen}()\right] = 1.$$

IND-CPA, IND-CCA, and SPR-CCA security of an OKEM are defined exactly as for the underlying KEM (see Figure 2), merely ignoring the encoded public key $\widehat{pk}$ output by KGen. We also adopt notions of public key and ciphertext uniformity of (O)KEMs from [GSV24].

Definition 2.6 (Public key uniformity)**.** *Let* O *be an OKEM. We measure the uniformity of the obfuscated public keys of length* ol *generated by* O.KGen *against an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{pk\text{-}unif}}_{\mathsf{O}}(\mathcal{A}) := 2 \cdot \Pr\left[\mathcal{A}(\widehat{pk}_b) = b \;\middle|\; \begin{array}{l} b \leftarrow\$ \{0,1\}, \widehat{pk}_0 \leftarrow\$ \{0,1\}^{\mathsf{ol}}, \\ (sk_1, pk_1, \widehat{pk}_1) \leftarrow\$ \mathsf{O.KGen}() \end{array}\right] - 1.$$

For an unbounded adversary $\mathcal{A}$*, we call the advantage* $\mathsf{Adv}^{\mathsf{pk\text{-}unif}}_{\mathsf{O}}(\mathcal{A})$ statistical*.*

Definition 2.7 (Ciphertext uniformity – strong or regular)**.** *Let* O *be an OKEM. We measure the* strong or regular ciphertext uniformity *of the obfuscated ciphertext of length* cl *generated by* O.Encap *against an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{atk\text{-}ctxt\text{-}unif}}_{\mathsf{O}}(\mathcal{A}) := 2 \cdot \Pr\left[\mathcal{A}(\boxed{sk,}\, pk, c_b) = b \;\middle|\; \begin{array}{l} b \leftarrow\$ \{0,1\}, c_0 \leftarrow\$ \{0,1\}^{\mathsf{cl}}, \\ (sk, pk, \widehat{pk}) \leftarrow\$ \mathsf{O.KGen}(), \\ (c_1, K_1) \leftarrow\$ \mathsf{O.Encap}(pk) \end{array}\right] - 1,$$

where $\mathsf{atk} \in \{\mathsf{strong}, \mathsf{reg}\}$ *and the code in the* $\boxed{\textit{dashed box}}$ *is only included for* $\mathsf{atk} = \mathsf{strong}$.

For an unbounded adversary $\mathcal{A}$*, we call the advantage* $\mathsf{Adv}^{\mathsf{atk\text{-}ctxt\text{-}unif}}_{\mathsf{O}}(\mathcal{A})$ statistical*.*

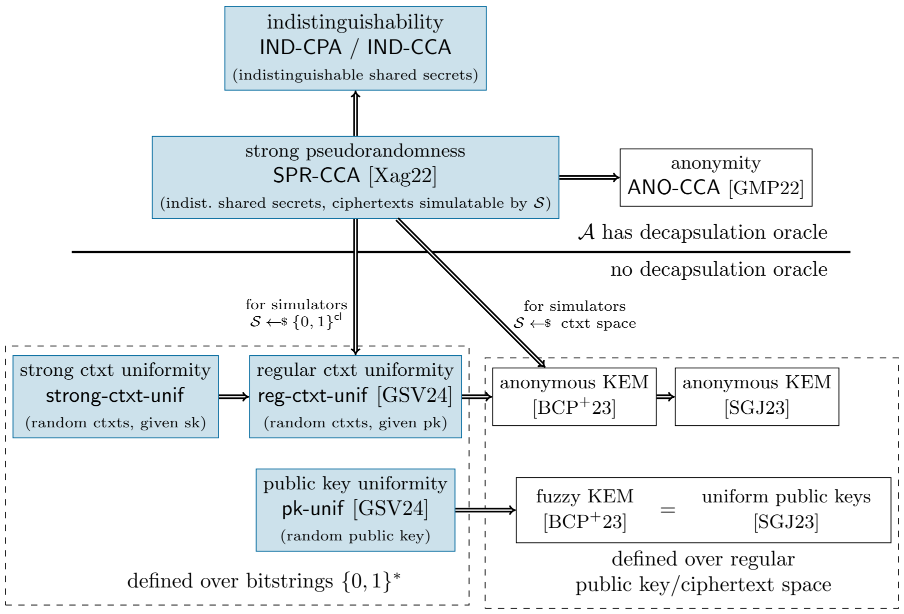


Figure 3: Relations between anonymity and uniformity notions for KEMs. Arrows $\Longrightarrow$ indicate strict implications. Properties shaded in blue are used in this paper.

Note that, when paired with an encoding function like Elligator2 [BHKL13], a Diffie–Hellman-based KEM over a prime-order curve (e.g., DHKEM(P-256) as defined in HPKE [BBLW22]) is an OKEM with statistical strong ciphertext uniformity and statistical public key uniformity (see Appendix D). Additionally, ML-KEM paired with the Kemeleon encoding is an OKEM with (computational) regular ciphertext uniformity and (computational) public key uniformity, as proven in [GSV24, §2.4]. Finally, Saber [DKRV18] and FrodoKEM [BCD+16] have the same properties as ML-KEM with Kemeleon and are "natural" OKEMs, i.e., KEMs where no encoding step is necessary in order to achieve obfuscation. This is because both KEMs, based on Module Learning-with-Rounding (MLWR) and unstructured Learning-with-Errors (LWE) respectively, use power-of-two moduli, making their public keys and ciphertexts pack perfectly into bitstrings. Their IND-CCA and SPR-CCA properties are stated in [Xag22, GMP22, MX23].[1]

The notions of strong pseudorandomness, public key uniformity, and (strong and regular) ciphertext uniformity are related to existing notions of anonymity and uniformity for KEMs in the literature, summarized in Figure 3. We elaborate on these relations in Appendix B, and note that all related notions are weaker than or equivalent to one of the properties we use here. The stronger notions of anonymity and uniformity are required for our later applications (see Sections 4 and 5). We focus on the above definitions of ciphertext and public key uniformity (rather than existing definitions of uniformity), because we require uniformity properties defined over the space of bitstrings $\{0,1\}^*$ rather than the space of public keys/ciphertexts as used in some prior works. Although *regular* ciphertext uniformity is implied by SPR-CCA (when SPR-CCA is defined with respect to a simulator that outputs

[1]Specifically, [GMP22] shows that FrodoKEM achieves computational ANO-CCA, and [Xag22] suggests that it also achieves computational SPR-CCA.

uniformly random bitstrings), we consider it separately because in some results we only require the weaker assumption.

3 OEINC: An OKEM Combiner

We now present our OKEM combiner and prove the necessary security properties for use in our hybrid obfuscated key exchange protocol (Section 4).

Shortcomings of existing combiners. A KEM combiner merges two ingredient KEMs into a single, hybrid KEM such that security of the hybrid is maintained as long as one of the ingredient KEMs remains secure. The main focus of prior work analyzing KEM combiners [GHP18, BBF$^+$19] has been on achieving hybrid IND-CCA security: an attacker should not be able to learn the secret key by breaking only one of the underlying KEMs. Although this is a natural starting point, some applications such as broadcast encryption, anonymous credential systems, and auction protocols, require anonymity properties not implied by IND-CCA security [GMP22]. Likewise, for our use case of constructing a hybrid obfuscated key exchange, we require that the combined KEM has *hybrid obfuscation* properties, to ensure that an adversary cannot distinguish the key exchange transcript from random by breaking obfuscation of only one of the underlying primitives.

Prior work on KEM combiners constructed *parallel combiners*: the combined public keys, secret keys, and ciphertexts are the concatenation of the ingredient KEM's public keys, secret keys, and ciphertexts, respectively (perhaps with an added MAC). The combined KEM's encapsulation and decapsulation routines first execute the underlying KEM's encapsulation and decapsulation routines and then join the resulting shared secrets using a *split-key PRF* [GHP18] to derive the final shared secret. Figure 1(a) illustrates a parallel combiner.

These existing combiners, however, do not provide hybrid obfuscation. Specifically, if an ingredient KEM's ciphertext is distinguishable from random, then its concatenation with another ciphertext is trivially distinguishable from random as well. For example, since the uniformity of ML-KEM ciphertexts relies on the module LWE assumption, breaking this assumption would be sufficient to violate the ciphertext uniformity of any parallel combiner using ML-KEM, such as Xyber [WW24] or X-Wing [BCD$^+$24]. Furthermore, it is not obvious how to fix this: if one were to encrypt each component ciphertext under a key from the respective other ingredient KEM, the result is impossible to decapsulate. Therefore, we require a more thoughtful construction.

The OEINC construction. Our OKEM combiner's encapsulation function operates sequentially, rather than in parallel. This new method, depicted in Figure 1(b), operates as follows. The first (*outer*) ingredient OKEM is run, and (a key derived from) its shared secret is used to encrypt the second (*inner*) ingredient OKEM's ciphertext. The final ciphertext is the outer ciphertext concatenated with the encrypted inner ciphertext. The shared secrets are combined with a split-key PRF. We call this the *Outer-Encrypts-Inner Nested Combiner* (OEINC).

The ciphertext uniformity of this new scheme relies on the security of the encryption scheme and the ciphertext uniformity of one of the two ingredient OKEMs. We use the OKEM with the stronger ciphertext uniformity property as the outer OKEM. When the outer OKEM has statistical strong ciphertext uniformity, e.g., as in DHKEM combined with Elligator2 (Appendix D), we find that the resulting combiner achieves hybrid security for the relevant notions.

```
$\underline{\mathsf{KGen}():}$
$(sk_{out}, pk_{out}, \widehat{pk}_{out}) \leftarrow\!\!\$ \ \mathsf{outOKEM.KGen}()$
$(sk_{in}, pk_{in}, \widehat{pk}_{in}) \leftarrow\!\!\$ \ \mathsf{inOKEM.KGen}()$
return $((sk_{out}, sk_{in}), (pk_{out}, pk_{in}), \widehat{pk}_{out} \| \widehat{pk}_{in})$

$\underline{\mathsf{Encap}(pk):}$
$(pk_{out}, pk_{in}) \leftarrow pk$
$(K_{out}, c_{out}) \leftarrow\!\!\$ \ \mathsf{outOKEM.Encap}(pk_{out})$
$(K_{oe}, K_{ok}) \leftarrow \mathsf{G}(K_{out})$ // keys for enc. & key deriv.
$(K_{in}, c_{in}) \leftarrow\!\!\$ \ \mathsf{inOKEM.Encap}(pk_{in})$
$c'_{in} \leftarrow \mathsf{SE.Enc}(K_{oe}, c_{in})$
$c \leftarrow c_{out} \| c'_{in}$
$K \leftarrow \mathsf{W}(K_{ok}, K_{in}, c)$
return $(K, c)$

$\underline{\mathsf{DecodePk}(pk):}$
$\widehat{pk}_{out} \| \widehat{pk}_{in} \leftarrow \widehat{pk}$
$pk_{out} \leftarrow \mathsf{outOKEM.DecodePk}(\widehat{pk}_{out})$
$pk_{in} \leftarrow \mathsf{inOKEM.DecodePk}(\widehat{pk}_{in})$
return $(pk_{out}, pk_{in})$

$\underline{\mathsf{Decap}(sk, c):}$
$(sk_{out}, sk_{in}) \leftarrow sk$
$c_{out} \| c'_{in} \leftarrow c$ // ciphertexts are fixed-length
$K_{out} \leftarrow \mathsf{outOKEM.Decap}(sk_{out}, c_{out})$
$(K_{oe}, K_{ok}) \leftarrow \mathsf{G}(K_{out})$
$c_{in} \leftarrow \mathsf{SE.Dec}(K_{oe}, c'_{in})$
$K_{in} \leftarrow \mathsf{inOKEM.Decap}(sk_{in}, c_{in})$
$K \leftarrow \mathsf{W}(K_{ok}, K_{in}, c)$
return $K$
```

Figure 4: Our OKEM combiner $\mathsf{OEINC}[\mathsf{outOKEM}, \mathsf{inOKEM}, \mathsf{SE}, \mathsf{G}, \mathsf{W}]$ for two OKEMs outOKEM and inOKEM, SE a length-preserving symmetric encryption scheme, G a PRG, and W a split-key PRF.

Our detailed combiner construction is given in Figure 4. Its building blocks include a length-preserving, OT-IND\$-secure symmetric encryption scheme SE with key space $\mathcal{K} = \{0,1\}^{\mathsf{kl}_{\mathsf{SE}}}$ and message length $\mathsf{cl}_{\mathsf{inOKEM}}$, a function $\mathsf{G}\colon \{0,1\}^{\mathsf{kl}} \to \{0,1\}^{\mathsf{kl}+\mathsf{kl}_{\mathsf{SE}}}$ assumed to be a pseudorandom generator, and a split-key PRF W.

3.1 Overview of Security

We now prove that OEINC satisfies IND-CPA[2], IND-CCA, SPR-CCA, ciphertext uniformity (ctxt-unif), and public key uniformity (pk-unif), given some properties of the underlying OKEMs. More specifically, for OEINC instantiated with an outer OKEM outOKEM and an inner OKEM inOKEM:

- IND-CPA: If either outOKEM or inOKEM is IND-CPA, then the combined OKEM is IND-CPA.
- IND-CCA: If either outOKEM or inOKEM is IND-CCA, then the combined OKEM is IND-CCA.
- SPR-CCA: If either (1) outOKEM is SPR-CCA, or (2) inOKEM is SPR-CCA and outOKEM is strong-ctxt-unif, then the combined OKEM is SPR-CCA.
- Ciphertext uniformity: If either (1) outOKEM is reg-ctxt-unif and outOKEM is IND-CCA, or (2) outOKEM is reg-ctxt-unif and inOKEM is reg-ctxt-unif, then the combined OKEM is reg-ctxt-unif.
- Public key uniformity: If both outOKEM and inOKEM are pk-unif, then the combined OKEM is pk-unif.

[2]We show IND-CPA separately from IND-CCA since the weaker notion is sufficient for some applications, e.g., password-authenticated key exchange (Section 5), and often comes with substantial performance benefits.

In particular, when $\mathsf{outOKEM}$ is statistically $\mathsf{strong\text{-}ctxt\text{-}unif}$, the combined OKEM has $\mathsf{IND\text{-}CPA}$, $\mathsf{IND\text{-}CCA}$, $\mathsf{SPR\text{-}CCA}$, and $\mathsf{reg\text{-}ctxt\text{-}unif}$ properties that are the stronger of the two underlying OKEMs.

Public key uniformity stands out because it requires *both* underlying OKEMs to be $\mathsf{pk\text{-}unif}$. While this limitation appears to be inherent to the primitive (when only the public key is known, there is no key material with which to obfuscate it), it is not a deal breaker for deployment. Our key exchange protocol in Section 4 does not require public key uniformity at all, since it encrypts ephemeral public keys using intermediate secrets derived from static keys. And where public key uniformity is necessary, e.g., in password-authenticated key exchange in Section 5, we show that it is possible to construct lattice-based OKEMs with statistical public key uniformity. This comes at the cost of significantly larger public keys and ciphertexts compared to, e.g., $\mathsf{ML\text{-}KEM}$ with $\mathsf{Kemeleon}$.

3.2 $\mathsf{IND\text{-}CPA}$ and $\mathsf{IND\text{-}CCA}$ Security

Intuitively, $\mathsf{IND\text{-}CPA}$ and $\mathsf{IND\text{-}CCA}$ security of OEINC should naturally hold, since the final shared secret is derived from the two underlying shared secrets using a split-key PRF. Importantly, OEINC does not use the same secret for encrypting ciphertexts (K_{oe}) as for key derivation (K_{ok}). The straightforward proof is deferred to Appendix C.1.

Theorem 3.1 ($\mathsf{IND\text{-}CPA}$ / $\mathsf{IND\text{-}CCA}$ security of OEINC). *Let* $\mathsf{OEINC} = \mathsf{OEINC}[\mathsf{outOKEM}, \mathsf{inOKEM}, \mathsf{SE}, \mathsf{G}, \mathsf{W}]$ *be a combined OKEM as defined in Figure 4. Then for any adversary* $\mathcal{A}$ *against the* $\mathsf{IND\text{-}atk}$ *security of* OEINC*, for* $\mathsf{atk} \in \{\mathsf{CPA}, \mathsf{CCA}\}$*, we give algorithms* $\mathcal{B}_1$*,* $\mathcal{B}_2$*,* $\mathcal{B}_3$*,* $\mathcal{C}_1$*,* $\mathcal{C}_2$ *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}atk}}_{\mathsf{OEINC}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{IND\text{-}atk}}_{\mathsf{outOKEM}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{PRG}}_{\mathsf{G}}(\mathcal{B}_2) + \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{W},1}(\mathcal{B}_3),$$

and

$$\mathsf{Adv}^{\mathsf{IND\text{-}atk}}_{\mathsf{OEINC}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{IND\text{-}atk}}_{\mathsf{inOKEM}}(\mathcal{C}_1) + \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{W},2}(\mathcal{C}_2).$$

3.3 $\mathsf{SPR\text{-}CCA}$ Security

Consider the case where the underlying computational assumption of inOKEM is broken. Then, assuming the $\mathsf{SPR\text{-}CCA}$ security of the outer OKEM holds, this implies that the outer ciphertext is uniformly random and the inner ciphertext is encrypted (using an $\mathsf{OT\text{-}IND\$}$ symmetric encryption scheme) with a random key, and thus also pseudorandom. The final shared secret is also random since the split-key PRF W takes as input the shared secret of the outer OKEM.

Now, consider the case where the underlying computational assumption of $\mathsf{outOKEM}$ is broken, i.e., $\mathsf{IND\text{-}CCA}$ and $\mathsf{SPR\text{-}CCA}$ no longer hold (note that strong ciphertext uniformity cannot fail when it is statistical). Then the outer ciphertext is still uniformly random, and the inner one is a keyed permutation of a uniformly random value, and thus still random even though the encryption key may be compromised. The shared secret is also uniformly random, since the split-key PRF W still consumes the shared secret of the inner OKEM.

Theorem 3.2 ($\mathsf{SPR\text{-}CCA}$ Security of OEINC). *Let* $\mathsf{OEINC} = \mathsf{OEINC}[\mathsf{outOKEM}, \mathsf{inOKEM}, \mathsf{SE}, \mathsf{G}, \mathsf{W}]$ *be a combined OKEM as defined in Figure 4. Then (stated informally)* OEINC *has strong pseudorandomness under chosen-ciphertext attack* ($\mathsf{SPR\text{-}CCA}$) *if either (I) the outer KEM* $\mathsf{outOKEM}$ *is* $\mathsf{SPR\text{-}CCA}$*-secure,* SE *is* $\mathsf{OT\text{-}IND\$}$*-secure,* G *is a PRG, and* W *is a PRF, or*

(II) inOKEM *is* $\mathsf{SPR\text{-}CCA}$*-secure,* $\mathsf{outOKEM}$ *has strong ciphertext uniformity, and* W *is a PRF. More precisely, for any adversary* $\mathcal{A}$ *against the* $\mathsf{SPR\text{-}CCA}$ *security of* OEINC*, we give algorithms* $\mathcal{B}_1$*–*$\mathcal{B}_4$ *and* $\mathcal{C}_1$*–*$\mathcal{C}_3$ *such that*

$$\mathsf{Adv}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{OEINC},\mathcal{S}_1}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{outOKEM},\mathcal{S}_{\mathsf{outOKEM}}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{PRG}}_{\mathsf{G}}(\mathcal{B}_2) + \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{W},1}(\mathcal{B}_3) + \mathsf{Adv}^{\mathsf{OT\text{-}IND\$}}_{\mathsf{SE}}(\mathcal{B}_4),$$

where $\mathcal{S}_1$ *is a simulator that returns a ciphertext* $\widetilde{c}_{\mathsf{outOKEM}} \| \widetilde{c}_{\mathsf{inOKEM}}$ *where* $\widetilde{c}_{\mathsf{outOKEM}} \leftarrow\!\$\ \mathcal{S}_{\mathsf{outOKEM}}$ *and* $\widetilde{c}_{\mathsf{inOKEM}} \leftarrow\!\$\ \{0,1\}^{\mathsf{cl}_{\mathsf{inOKEM}}}$*, and*

$$\mathsf{Adv}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{OEINC},\mathcal{S}_2}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{inOKEM},\mathcal{S}_{\mathsf{inOKEM}}}(\mathcal{C}_1) + \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{W},2}(\mathcal{C}_2) + \mathsf{Adv}^{\mathsf{strong\text{-}ctxt\text{-}unif}}_{\mathsf{outOKEM}}(\mathcal{C}_3),$$

where $\mathcal{S}_{\mathsf{inOKEM}}$ *and* $\mathcal{S}_2$ *sample ciphertexts uniformly at random from* $\{0,1\}^{\mathsf{cl}_{\mathsf{inOKEM}}}$ *and* $\widetilde{c} \leftarrow\!\$\ \{0,1\}^{\mathsf{cl}_{\mathsf{inOKEM}}+\mathsf{cl}_{\mathsf{outOKEM}}}$*, respectively.*

We note that when $\mathcal{S}_{\mathsf{outOKEM}}$ *is the simulator that samples ciphertexts uniformly at random from* $\{0,1\}^{\mathsf{cl}_{\mathsf{outOKEM}}}$*, then* $\mathcal{S}_1 = \mathcal{S}_2$*.*[3]

Proof. **Game 0.** We start with the security game for $\mathsf{SPR\text{-}CCA}$ ($\mathsf{G}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{OEINC},\mathcal{S}_1}(\mathcal{A})$).

Case I.

In the first case, we reduce to $\mathsf{SPR\text{-}CCA}$ security of the outer OKEM, and proceed via a series of game hops.

Game I.0. In $\mathsf{G}_{\mathrm{I}.0}$, we replace (in the challenge encapsulation yielding K_1^* and c_1^*) the outer OKEM shared secret K_{out} and ciphertext c_{out} with a random key $\widetilde{K}_{out} \leftarrow\!\$\ \mathcal{K}_{\mathsf{outOKEM}}$ and simulated ciphertext $\widetilde{c}_{out} \leftarrow\!\$\ \mathcal{S}_{\mathsf{outOKEM}}$. We bound the adversary's difference in advantage by a reduction $\mathcal{B}_1$ to the $\mathsf{SPR\text{-}CCA}$ of $\mathsf{outOKEM}$ w.r.t the simulator $\mathcal{S}_{\mathsf{outOKEM}}$. $\mathcal{B}_1$ obtains the $\mathsf{SPR\text{-}CCA}$ challenge (pk, c^*, K^*) for $\mathsf{outOKEM}$ and simulates the game as follows for $\mathcal{A}$. It uses pk in place of pk_{out}. When $\mathcal{O}_{\mathsf{Decap}}$ is called, if $c_{out} = c^*$ (in line 17 of Figure 4) then $\mathcal{B}_1$ computes the remainder of Decap using K^* as K_{out}; else, if $c_{out} \neq c^*$ then $\mathcal{B}_1$ queries its $\mathsf{SPR\text{-}CCA}$ decapsulation oracle (for $\mathsf{outOKEM}$) and uses the response as K_{out}.

If (c^*, K^*) are real values then $\mathcal{B}_1$ exactly simulates G_0 to $\mathcal{A}$; else, $\mathcal{B}_1$ simulates $\mathsf{G}_{\mathrm{I}.0}$ to $\mathcal{A}$. Therefore:

$$\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_{\mathrm{I}.0}] \leq \mathsf{Adv}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{outOKEM},\mathcal{S}_{\mathsf{outOKEM}}}(\mathcal{B}_1).$$

Game I.1. In $\mathsf{G}_{\mathrm{I}.1}$, for the computation of the challenge shared secret K_1^*, we replace the output K_{oe}, K_{ok} of $\mathsf{G}(\widetilde{K}_{out})$ with uniformly random $\widetilde{K}_{oe}, \widetilde{K}_{ok}$, in particular for the challenge. We bound the difference in this step by a reduction $\mathcal{B}_2$ to the PRG security of G. The reduction uses its oracle in place of G, simulating either $\mathsf{G}_{\mathrm{I}.0}$ (if the G oracle output is real) or $\mathsf{G}_{\mathrm{I}.1}$ (if the G oracle output is random), giving:

$$\Pr[\mathsf{G}_{\mathrm{I}.0}] - \Pr[\mathsf{G}_{\mathrm{I}.1}] \leq \mathsf{Adv}^{\mathsf{PRG}}_{\mathsf{G}}(\mathcal{B}_2).$$

[3]Our result for $\mathsf{SPR\text{-}CCA}$ security is specific to the case where the inner KEM produces ciphertexts that look like random bitstrings. It can be generalized to the case where both outer and inner KEMs produce structured, simulatable ciphertexts (e.g., in many non-obfuscated KEMs) by assuming that SE is an ideal cipher, using a generalized definition of $\mathsf{strong\text{-}ctxt\text{-}unif}$ that captures indistinguishability from simulated ciphertexts, and adjusting the proof accordingly. Our focus here is on obfuscated KEMs with uniformly random ciphertexts.

```
$\underline{\mathsf{Encap}(pk)\text{:}}$
1  $(pk_{out}, pk_{in}) \leftarrow pk$
2  $(\widetilde{K}_{out}, \widetilde{c}_{out}) \leftarrow\!\$\ \mathcal{K}_{\mathsf{outOKEM}} \times \mathcal{S}_{\mathsf{outOKEM}}$  // $\mathsf{G}_{\mathrm{I}.0}$
3  $(\widetilde{K}_{oe}, \widetilde{K}_{ok}) \leftarrow\!\$\ \{0,1\}^{\mathsf{cl}_{\mathsf{inOKEM}}} \times \ldots$  // $\mathsf{G}_{\mathrm{I}.1}$
4  $(K_{in}, c_{in}) \leftarrow\!\$\ \mathsf{inOKEM.Encap}(pk_{in})$
5  $c'_{in} \leftarrow \mathsf{SE.Enc}(\widetilde{K}_{oe}, c_{in})$
6  $c \leftarrow c_{out} \| c'_{in}$
7  $K \leftarrow \widetilde{\mathsf{W}}(K_{in}, c)$  // $\mathsf{G}_{\mathrm{I}.2}$
8  return $(K, c)$
```

```
$\underline{\mathsf{Decap}(sk, c)\text{:}}$
9  $(sk_{out}, sk_{in}) \leftarrow sk$
10 $c_{out} \| c'_{in} \leftarrow c$
11 if $c_{out} \neq \widetilde{c}_{out}$ then $\ldots$ // handle regularly
   // else: use $\widetilde{K}_{oe}$, $\widetilde{K}_{ok}$
12 $c_{in} \leftarrow \mathsf{SE.Dec}(\widetilde{K}_{oe}, c'_{in})$
13 $K_{in} \leftarrow \mathsf{inOKEM.Decap}(sk_{in}, c_{in})$
14 $K \leftarrow \widetilde{\mathsf{W}}(K_{in}, c)$  // $\mathsf{G}_{\mathrm{I}.3}$: $K_{in} \leftarrow 0$
15 return $K$
```

Figure 5: Modifications in the game hops $\mathsf{G}_{\mathrm{I}.0}$–$\mathsf{G}_{\mathrm{I}.3}$ of the $\mathsf{SPR\text{-}CCA}$ proof, case (I), to the encapsulation and decapsulation algorithms of OEINC (Figure 2, line 9 resp. 15).

Game I.2. We now replace evaluations of $\mathsf{W}(\widetilde{K}_{ok}, \cdot, \cdot)$ with a random function $\widetilde{\mathsf{W}}(\cdot, \cdot)$. This in particular replaces the challenge shared secret K with a uniformly random value, independent of outputs of the decapsulation oracle since the third input c to W must be distinct from the challenge c^* for each query.

We bound this step by a reduction $\mathcal{B}_3$ to the split-key pseudorandomness of W, where $\mathcal{B}_3$ uses its oracle in place of calls to $\mathsf{W}(\widetilde{K}_{ok}, \cdot, \cdot)$. It follows that

$$\Pr[\mathsf{G}_{\mathrm{I}.1}] - \Pr[\mathsf{G}_{\mathrm{I}.2}] \leq \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{W},1}(\mathcal{B}_3).$$

Game I.3. Next, we rewrite the decapsulation oracle when queried with the challenge outer ciphertext $\widetilde{c}_{out}$ in a way that is unnoticeable to the adversary. First, we fix the K_{in} input to $\widetilde{\mathsf{W}}$ (Figure 5, line 14) to a zero string. Since K_{in} is deterministically derived from the second input $c = \widetilde{c}_{out} \| c'_{in}$ (as $\widetilde{K}_{oe}$ and sk_{in} are fixed), this does not change the distribution of K: decapsulating a ciphertext $c = \widetilde{c}_{out} \| c'_{in}$ will yield distinct, randomly sampled shared secrets for distinct c'_{in}. Then

$$\Pr[\mathsf{G}_{\mathrm{I}.2}] = \Pr[\mathsf{G}_{\mathrm{I}.3}].$$

From this point on, we no longer need to decapsulate the corresponding inner ciphertext (lines 12 and 13 of Figure 5 highlighted in gray), and hence also do not need to use $\widetilde{K}_{oe}$ anymore to decrypt c'_{in} when answering decapsulation queries on ciphertexts containing the challenge outer ciphertext $\widetilde{c}_{out}$.

Game I.4. Finally, in $\mathsf{G}_{\mathrm{I}.4}$ we replace the encrypted inner ciphertext c'_{in} of the challenge with random bits of length $\mathsf{cl}_{\mathsf{inOKEM}}$, by a reduction $\mathcal{B}_4$ to the $\mathsf{OT\text{-}IND\$}$ security of SE. $\mathcal{B}_4$, on input c_{in}, obtains a challenge ciphertext c^* and simulates the game for $\mathcal{A}$, replacing c'_{in} with c^* in the challenge ciphertext. Importantly, $\mathcal{B}_4$ does not need to know the encryption key $\widetilde{K}_{oe}$ to answer decapsulation queries on ciphertexts containing the challenge outer ciphertext $\widetilde{c}_{out}$, as per $\mathsf{G}_{\mathrm{I}.3}$.

Hence $\mathcal{B}_4$ exactly simulates $\mathsf{G}_{\mathrm{I}.3}$ or $\mathsf{G}_{\mathrm{I}.4}$ to $\mathcal{A}$. Therefore:

$$\Pr[\mathsf{G}_{\mathrm{I}.3}] - \Pr[\mathsf{G}_{\mathrm{I}.4}] \leq \mathsf{Adv}^{\mathsf{OT\text{-}IND\$}}_{\mathsf{SE}}(\mathcal{B}_4).$$

At this point, by $\mathsf{G}_{\mathrm{I}.0}$ and $\mathsf{G}_{\mathrm{I}.4}$, the ciphertext received by $\mathcal{A}$ is uniformly random and by $\mathsf{G}_{\mathrm{I}.2}$ the key is uniformly random. Therefore,

$$\mathsf{Adv}^{\mathsf{G}_{\mathrm{I}.4}}_{\mathsf{OEINC},\mathcal{S}_1}(\mathcal{A}) = 0.$$

Collecting the bounds yields the theorem statement.

Case II.

In the second case, we reduce to $\mathsf{SPR\text{-}CCA}$ security of inOKEM and $\mathsf{strong\text{-}ctxt\text{-}unif}$ security of $\mathsf{outOKEM}$.

Game II.0. In $\mathsf{G}_{\text{II}.0}$, we replace (in the challenge encapsulation yielding K_1^* and c_1^*) the inner OKEM shared secret K_{in} and ciphertext c_{in} with a random shared secret $\widetilde{K}_{in} \leftarrow_\$ \mathcal{K}_{\mathsf{inOKEM}}$ and random ciphertext $\widetilde{c}_{in} \leftarrow_\$ \{0,1\}^{\mathsf{cl}_{\mathsf{inOKEM}}}$. The adversary $\mathcal{A}$'s advantage is then bounded by a reduction $\mathcal{C}_1$ to the $\mathsf{SPR\text{-}CCA}$ security of inOKEM with respect to the simulator $\mathcal{S}_{\mathsf{inOKEM}}$ that samples ciphertexts uniformly at random from $\{0,1\}^{\mathsf{cl}_{\mathsf{inOKEM}}}$. $\mathcal{C}_1$ obtains an $\mathsf{SPR\text{-}CCA}$ challenge (pk, c^*, K^*) and simulates the game for $\mathcal{A}$ by using pk in place of pk_{in}. When $\mathcal{O}_{\mathsf{Decap}}$ is called, if $c_{in} = c^*$ (in line 20 of Figure 4), then $\mathcal{C}_1$ uses K^* in place of K_{in}; else (when $c_{in} \neq c^*$), $\mathcal{C}_1$ queries its $\mathsf{SPR\text{-}CCA}$ decapsulation oracle (for inOKEM) and uses the response as K_{in}.

If (c^*, K^*) are real then $\mathcal{C}_1$ exactly simulates G_0 to $\mathcal{A}$. Otherwise, $\mathcal{C}_1$ simulates $\mathsf{G}_{\text{II}.0}$ to $\mathcal{A}$. Thus,

$$\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_{\text{II}.0}] \leq \mathsf{Adv}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{inOKEM},\mathcal{S}_{\mathsf{inOKEM}}}(\mathcal{C}_1).$$

Game II.1. Next, we replace evaluations of $\mathsf{W}(\cdot, \widetilde{K}_{in}, \cdot)$ with a random function. This in particular replaces K (both in the challenge computation and in the decapsulation oracle) with a uniformly random value, independent of outputs of the decapsulation oracle since the third input c to W is distinct from the challenge c^* for each query. This step is bounded by a reduction $\mathcal{C}_2$ to the split-key pseudorandomness of W, where $\mathcal{C}_2$ uses its oracle in place of calls to $\mathsf{W}(\cdot, \widetilde{K}_{in}, \cdot)$. Since $\widetilde{K}_{in}$ is random by $\mathsf{G}_{\text{II}.0}$, it follows that

$$\Pr[\mathsf{G}_{\text{II}.0}] - \Pr[\mathsf{G}_{\text{II}.1}] \leq \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{W},2}(\mathcal{C}_2).$$

Game II.2. In $\mathsf{G}_{\text{II}.2}$, we replace (in the challenge encapsulation) the encrypted inner ciphertext c'_{in} with a random ciphertext in $\{0,1\}^{\mathsf{cl}_{\mathsf{inOKEM}}}$. This follows directly from the fact that $\mathsf{SE.Enc}(K_{oe}, \cdot)$ is a permutation over $\{0,1\}^{\mathsf{cl}_{\mathsf{inOKEM}}}$ and that $\widetilde{c}_{in}$ is a random bitstring (by $\mathsf{G}_{\text{II}.0}$). Therefore,

$$\Pr[\mathsf{G}_{\text{II}.1}] = \Pr[\mathsf{G}_{\text{II}.2}].$$

Game II.3. Now, the final shared secret K and the inner ciphertext are random (and independent of K_{out}), so it only remains to replace the outer ciphertext c_{out} with random. In $\mathsf{G}_{\text{II}.3}$, we replace c_{out} with a randomly sampled ciphertext from $\{0,1\}^{\mathsf{cl}_{\mathsf{outOKEM}}}$. We bound this change by a reduction $\mathcal{C}_3$ to the $\mathsf{strong\text{-}ctxt\text{-}unif}$ of $\mathsf{outOKEM}$. In particular, $\mathcal{C}_3$ simulates the game for $\mathcal{A}$ by using its challenge ciphertext c^* from the $\mathsf{strong\text{-}ctxt\text{-}unif}$ game in place of c_{out}. To answer $\mathcal{O}_{\mathsf{Decap}}$ queries, $\mathcal{C}_3$ uses the given sk from the $\mathsf{strong\text{-}ctxt\text{-}unif}$ challenge to simulate calls to $\mathsf{outOKEM.Decap}(sk_{out}, c_{out})$. Then, $\mathcal{C}_3$ exactly simulates $\mathsf{G}_{\text{II}.2}$ (if the $\mathsf{strong\text{-}ctxt\text{-}unif}$ challenge is real) or $\mathsf{G}_{\text{II}.3}$ (if the $\mathsf{strong\text{-}ctxt\text{-}unif}$ challenge is random) to $\mathcal{A}$. Therefore,

$$\Pr[\mathsf{G}_{\text{II}.2}] - \Pr[\mathsf{G}_{\text{II}.3}] \leq \mathsf{Adv}^{\mathsf{strong\text{-}ctxt\text{-}unif}}_{\mathsf{outOKEM}}(\mathcal{C}_3).$$

Now, each of K, c_{out}, and c'_{in} are random. Hence, the adversary $\mathcal{A}$ can only guess the challenge bit b:

$$\mathsf{Adv}^{\mathsf{G}_{\mathrm{II}.3}}_{\mathsf{OEINC},\mathcal{S}_2}(\mathcal{A}) = 0.$$

Collecting the bounds yields the theorem statement. □

3.4 Ciphertext Uniformity

A hybrid ciphertext uniformity guarantee holds if the outer OKEM has statistical (regular) ciphertext uniformity. When the computational assumption underlying inOKEM is broken and the assumption underlying outOKEM holds, the outer KEM ciphertext is still uniformly random by its ciphertext uniformity, and the shared secret derived from outOKEM is used to encrypt the inner ciphertext, thus retaining its uniformity. Notably, the encryption step should not provide any additional information about the inner ciphertext; any use of authentication (such as AE or a MAC) would provide an oracle to the adversary distinguishing inner ciphertexts from random. On the other hand, if the outOKEM assumption is broken, then the encryption key no longer provides any guarantee, so ciphertext uniformity relies on that of both the outer and inner ciphertexts.

Theorem 3.3 (reg-ctxt-unif of OEINC)**.** *Let* $\mathsf{OEINC} = \mathsf{OEINC}[\mathsf{outOKEM}, \mathsf{inOKEM}, \mathsf{SE}, \mathsf{G}, \mathsf{W}]$ *be a combined OKEM as defined in Figure 4. For any adversary* $\mathcal{A}$ *against the regular ciphertext uniformity (Definition 2.7) of* OEINC*, we give algorithms* $\mathcal{B}_1$*–*$\mathcal{B}_4$ *and* $\mathcal{C}_1$*–*$\mathcal{C}_2$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\text{reg-ctxt-unif}}_{\mathsf{OKEM}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{outOKEM}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{PRG}}_{\mathsf{G}}(\mathcal{B}_2) + \mathsf{Adv}^{\mathsf{OT\text{-}IND\$}}_{\mathsf{SE}}(\mathcal{B}_3)\\ + \mathsf{Adv}^{\text{reg-ctxt-unif}}_{\mathsf{outOKEM}}(\mathcal{B}_4),\end{aligned}$$

and

$$\mathsf{Adv}^{\text{reg-ctxt-unif}}_{\mathsf{OKEM}}(\mathcal{A}) \leq \mathsf{Adv}^{\text{reg-ctxt-unif}}_{\mathsf{inOKEM}}(\mathcal{C}_1) + \mathsf{Adv}^{\text{reg-ctxt-unif}}_{\mathsf{outOKEM}}(\mathcal{C}_2).$$

Proof. **Game 0.** We proceed via a series of game hops, starting with the security game for reg-ctxt-unif, $\mathsf{G}^{\text{reg-ctxt-unif}}_{\mathsf{OEINC}}(\mathcal{A})$.

Case I.

In the first case, we begin by reducing to IND-CPA security and reg-ctxt-unif of the outer KEM.

Game I.0. In $\mathsf{G}_{\mathrm{I}.0}$, we replace the outer KEM shared secret K_{out} with a uniformly random shared secret $\widetilde{K}_{out}$. All values derived from K_{out} use the random value $\widetilde{K}_{out}$.

We bound the difference in this step by a reduction $\mathcal{B}_1$ to the IND-CPA security of outOKEM. $\mathcal{B}_1$ obtains the IND-CPA challenge (pk, c^*, K^*) and simulates the game for $\mathcal{A}$ by using pk, c^*, and K^* as pk_{out}, c_{out}, and K_{out}, respectively. (Note that there are no decapsulation queries to be handled for reg-ctxt-unif security.) If K^* is a real KEM shared secret then $\mathcal{B}_1$ has exactly simulated G_0 to $\mathcal{A}$; else, if K^* is random, then $\mathcal{B}_1$ has exactly simulated $\mathsf{G}_{\mathrm{I}.0}$ to $\mathcal{A}$. Therefore:

$$\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_{\mathrm{I}.0}] \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{outOKEM}}(\mathcal{B}_1).$$

Game I.1. In $\mathsf{G}_{\mathrm{I}.1}$, we replace (K_{oe}, K_{ok}) with uniformly random values $\widetilde{K}_{oe}$, and $\widetilde{K}_{ok}$. We bound the difference in this step by a reduction $\mathcal{B}_2$ to the PRG security of G. The reduction uses its input in place of K_{oe}, K_{ok}. Since K_{out} is random by $\mathsf{G}_{\mathrm{I}.0}$, $\mathcal{B}_2$ simulates either $\mathsf{G}_{\mathrm{I}.0}$ or $\mathsf{G}_{\mathrm{I}.1}$, giving:

$$\Pr[\mathsf{G}_{\mathrm{I}.0}] - \Pr[\mathsf{G}_{\mathrm{I}.1}] \leq \mathsf{Adv}_{\mathsf{G}}^{\mathsf{PRG}}(\mathcal{B}_2).$$

Game I.2. In $\mathsf{G}_{\mathrm{I}.2}$, we replace the encrypted inner ciphertext c'_{in} of the challenge with a random ciphertext in $\{0,1\}^{\mathsf{cl}_{\mathsf{inOKEM}}}$. Note that K_{oe} is random by $\mathsf{G}_{\mathrm{I}.1}$, so this change can be bounded by a reduction $\mathcal{B}_3$ to the $\mathsf{OT\text{-}IND\$}$ security of SE. $\mathcal{B}_3$ obtains a challenge ciphertext c^* and simulates the game for $\mathcal{A}$, replacing c'_{in} with c^*. It follows that:

$$\Pr[\mathsf{G}_{\mathrm{I}.1}] - \Pr[\mathsf{G}_{\mathrm{I}.2}] \leq \mathsf{Adv}_{\mathsf{SE}}^{\mathsf{OT\text{-}IND\$}}(\mathcal{B}_3).$$

Game I.3. Finally, in $\mathsf{G}_{\mathrm{I}.3}$, we replace the outer KEM ciphertext c_{out} with a random ciphertext in $\{0,1\}^{\mathsf{cl}_{\mathsf{outOKEM}}}$. We bound the difference by a reduction $\mathcal{B}_4$ to the $\mathsf{reg\text{-}ctxt\text{-}unif}$ of $\mathsf{outOKEM}$. $\mathcal{B}_4$ uses its ciphertext challenge c_b in place of c_{out} and pk for pk_{out}; note that K_{out} isn't used anymore at this point. Exactly simulating $\mathsf{G}_{\mathrm{I}.2}$ if c_b is real, and simulating $\mathsf{G}_{\mathrm{I}.3}$ is c_b is random, we have that:

$$\Pr[\mathsf{G}_{\mathrm{I}.2}] - \Pr[\mathsf{G}_{\mathrm{I}.3}] \leq \mathsf{Adv}_{\mathsf{outOKEM}}^{\mathsf{reg\text{-}ctxt\text{-}unif}}(\mathcal{B}_4).$$

Now, the ciphertext received by $\mathcal{B}$ consists of a random $c_{out} \in \{0,1\}^{\mathsf{cl}_{\mathsf{outOKEM}}}$ by $\mathsf{G}_{\mathrm{I}.3}$, and a random $c'_{in} \in \{0,1\}^{\mathsf{cl}_{\mathsf{inOKEM}}}$ by $\mathsf{G}_{\mathrm{I}.2}$. Hence, $\mathcal{A}$ has no better chance than guessing the challenge bit b:

$$\mathsf{Adv}_{\mathsf{OEINC}}^{\mathsf{G}_{\mathrm{I}.3}}(\mathcal{A}) = 0.$$

Case II.

This case reduces to $\mathsf{reg\text{-}ctxt\text{-}unif}$ of inOKEM and $\mathsf{outOKEM}$.

Game II.0. In $\mathsf{G}_{\mathrm{II}.0}$, we replace the inner ciphertext c_{in} with a random ciphertext in $\{0,1\}^{\mathsf{cl}_{\mathsf{inOKEM}}}$. We bound the difference by a reduction $\mathcal{C}_1$ to the $\mathsf{reg\text{-}ctxt\text{-}unif}$ of inOKEM. $\mathcal{C}_1$ uses its ciphertext challenge c_b in place of c_{in}, and pk, K in place of pk_{in}, K_{in}. This simulates G_0 if c_b is real, and $\mathsf{G}_{\mathrm{II}.0}$ if c_b is random:

$$\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_{\mathrm{II}.0}] \leq \mathsf{Adv}_{\mathsf{inOKEM}}^{\mathsf{reg\text{-}ctxt\text{-}unif}}(\mathcal{C}_1).$$

Game II.1. Next, we replace c'_{in} with a random ciphertext in $\{0,1\}^{\mathsf{cl}_{\mathsf{inOKEM}}}$ in $\mathsf{G}_{\mathrm{II}.1}$. c_{in} is random by $\mathsf{G}_{\mathrm{II}.0}$, so it follows directly from the fact that $\mathsf{SE.Enc}(K_{oe}, \cdot)$ is a permutation over $\{0,1\}^{\mathsf{cl}_{\mathsf{inOKEM}}}$, for a fixed key K_{oe}, that:

$$\Pr[\mathsf{G}_{\mathrm{II}.0}] = \Pr[\mathsf{G}_{\mathrm{II}.1}].$$

Game II.2. Lastly, in $\mathsf{G}_{\mathrm{II}.2}$, we replace the outer ciphertext c_{out} with a random ciphertext in $\{0,1\}^{\mathsf{cl}_{\mathsf{outOKEM}}}$. We bound this by a reduction $\mathcal{C}_2$ to $\mathsf{reg\text{-}ctxt\text{-}unif}$ of $\mathsf{outOKEM}$. $\mathcal{C}_2$ simulates either $\mathsf{G}_{\text{II}.1}$ or $\mathsf{G}_{\text{II}.2}$ by using its challenge ciphertext in place of c_{out} (and pk, K in place of pk_{out}, K_{out}). Therefore:

$$\Pr[\mathsf{G}_{\text{II}.1}] - \Pr[\mathsf{G}_{\text{II}.2}] \leq \mathsf{Adv}^{\text{reg-ctxt-unif}}_{\text{outOKEM}}(\mathcal{C}_2).$$

At this point, both c'_{in} and c_{out} are random bitstrings. Thus, $\mathcal{A}$ can only guess the challenge bit b:

$$\mathsf{Adv}^{\mathsf{G}_{\text{II}.2}}_{\mathsf{OEINC}}(\mathcal{A}) = 0.$$

Collecting the bounds yields the theorem statement. □

3.5 Public Key Uniformity

Finally, OEINC only has public key uniformity as long as *both* underlying OKEMs retain their public key uniformity. Since no shared key material is available when distributing a public key, there is little that would help in obfuscating a public key if the underlying assumption is broken. Thus, if one of the schemes does not have statistical public key uniformity, there does not appear to be a way to construct a scheme with hybrid public key uniformity, barring some additional PKI-like infrastructure. OEINC does achieve public key uniformity conditioned on the public key uniformity of *both* underlying schemes:

Theorem 3.4 (pk-unif of OEINC). *Let* $\mathsf{OEINC} = \mathsf{OEINC}[\mathsf{outOKEM}, \mathsf{inOKEM}, \mathsf{SE}, \mathsf{G}, \mathsf{W}]$ *be a combined OKEM as defined in Figure 4. For any adversary* $\mathcal{A}$ *against the public key uniformity (Definition 2.6) of* OEINC*, we give algorithms* $\mathcal{B}_1$*,* $\mathcal{B}_2$ *such that*

$$\mathsf{Adv}^{\text{pk-unif}}_{\mathsf{OEINC}}(\mathcal{A}) \leq \mathsf{Adv}^{\text{pk-unif}}_{\mathsf{outOKEM}}(\mathcal{B}_1) + \mathsf{Adv}^{\text{pk-unif}}_{\mathsf{inOKEM}}(\mathcal{B}_2)$$

We defer the proof of public key uniformity to Appendix C.2.

3.6 Instantiating the Combiner

Following our results for OEINC, we now know that we can construct an OKEM with hybrid guarantees when the outer OKEM has statistical strong ciphertext uniformity. Previous work [GSV24] constructed an MLWE-based OKEM, ML-Kemeleon, from ML-KEM and Kemeleon encodings, notably whose ciphertext (and public key) uniformity depends on the MLWE assumption, i.e., it is not statistical. Similarly, Saber [DKRV18] and FrodoKEM [BCD+16] achieve uniformity from MLWR and LWE assumptions, respectively. These are natural choices for the inner OKEM. As for the outer OKEM, we can construct a (classically-secure) OKEM using DHKEM [BBLW22] and Elligator2 [BHKL13] or Elligator^2 [Tib14] encodings. We provide the details of this construction, denoted DHKEM-Ell2, and related security properties in Appendix D. Importantly, Elligator-style encodings do not rely on any underlying computational assumptions for uniformity guarantees, thus giving the resulting OKEM *statistical* strong ciphertext uniformity. Both ML-Kemeleon and DHKEM-Ell2 are SPR-CCA-secure with respect to simulators that output random bitstrings. The symmetric encryption SE can be instantiated with XOR (noting that the one-time pad gives an OT-IND\$ guarantee, and is a permutation over bitstrings for a fixed key). Therefore, instantiating $\mathsf{OEINC} = \mathsf{OEINC}[\mathsf{DHKEM\text{-}Ell2}, \mathsf{ML\text{-}Kemeleon}, \oplus, \mathsf{G}, \mathsf{W}]$ for any reasonable choice of PRG G and split-key PRF W (e.g., see [GHP18, BBF+19]) provides the desired hybrid guarantees.

We now explore alternative options for instantiating the combined OKEM, omitting any further discussion of SE, G, and W, as they can be instantiated with $\oplus$, any secure PRG, and

Table 1: Security properties of combiners instantiated with different input (O)KEMs. Underlying computational assumptions are given in parentheses. Dots in the table indicate: (at least) one of the underlying assumptions must hold, i.e., hybrid guarantee (●); the assumption of the outer KEM must hold (⭘); both underlying assumptions must hold (◎); or the property does not hold (–). For SPR-CCA, subscripts indicate whether the respective simulator samples bitstrings ($) or uniform ciphertexts from the ciphertext space ($\mathcal{C}$).

Combined KEM		IND-CCA	SPR-CCA	reg-ctxt-unif	pk-unif
OEINC **Outer and Inner KEM**					
DHKEM-Ell2(GapDH)	ML-Kemeleon(MLWE) Saber (MLWR) FrodoKEM (LWE)	●	●$_{\$}$	●	◎
ML-Kemeleon(MLWE)	DHKEM-Ell2(GapDH)	●	⭘$_{\$}$	⭘	◎
DHKEM-Ell2(GapDH)	ML-KEM(MLWE)	●	●$_{\$}$	⭘	–
DHKEM(GapDH)	ML-KEM(MLWE)	●	⭘$_{\mathcal{C}}$	–	–
X-Wing [BCD+24] (MLWE, GapDH)		●	◎$_{\mathcal{C}}$	–	–
Parallel Combiner [GHP18, BBF+19] ML-Kemeleon + DHKEM-Ell2 (MLWE, GapDH)		●	◎$_{\$}$	◎	◎

any split-key PRF, respectively. Table 1 summarizes the properties of each of the following variants, and compares them with the security properties provided by X-Wing [BCD+24] (which does not use obfuscated KEMs) and the generic parallel combiner [GHP18, BBF+19] instantiated with two obfuscated KEMs.

Non-obfuscated KEMs. It should be noted that the OEINC construction is not restricted to only OKEMs. Ignoring the step that encodes public keys, one can use any KEM as an input to the combiner. It is then interesting to ask what properties are maintained by the combiner in such cases. Of course, using any two IND-CCA secure KEMs (e.g., ML-KEM and DHKEM) results in an IND-CCA combined KEM. The remaining properties depend on the input KEMs. For example, using ML-KEM as the inner KEM and DHKEM as the outer KEM, the combined KEM would not achieve public key or ciphertext uniformity; however, it does achieve SPR-CCA security with respect to a simulator that outputs structured ciphertexts (i.e., DH values concatenated with random bitstrings resulting from encrypting ML-KEM ciphertexts), assuming that the outer KEM retains its SPR-CCA security. It is important to note that if the assumption underlying DHKEM is broken, then this construction no longer achieves SPR-CCA security, regardless of the security of the inner KEM.

Outer OKEM and inner KEM. Alternatively, one might ask what happens if only the outer KEM is obfuscated and the inner KEM is not. To illustrate, we consider the combination of DHKEM-Ell2 as the outer KEM and ML-KEM as the inner KEM. It is easy to see that IND-CCA security is maintained as a hybrid guarantee and that public key uniformity is no longer achieved. The combined KEM in this case would still have SPR-CCA security with respect to a simulator that outputs random bitstrings, as long as at least one of the input KEMs is SPR-CCA-secure (i.e., the hybrid guarantee is maintained). The notable difference is that there is no hybrid guarantee for ciphertext uniformity, which now relies on the IND-CPA security and ciphertext uniformity of DHKEM-Ell2 (since ML-KEM has no ciphertext uniformity).

Outer OKEM without strong-ctxt-unif. Finally, we ask what happens if we use ML-Kemeleon as the outer KEM and DHKEM-Ell2 as the inner KEM (swapping the order of our initial proposed instantiation). The (hybrid) IND-CCA and (non-hybrid) public key uniformity guarantees remain the same as in the original case. SPR-CCA is not a hybrid guarantee anymore and now depends on SPR-CCA security of ML-Kemeleon, since ML-Kemeleon does not have statistical strong ciphertext uniformity. Nonetheless, the SPR-CCA guarantee still holds with respect to a simulator that outputs random bitstrings. Similarly, ciphertext uniformity requires now that the ciphertext uniformity (and IND-CPA security) of ML-Kemeleon is maintained (i.e., no hybrid guarantee).

4 A Hybrid Obfuscated Key Exchange Protocol

We now present `Drivel`, a revised version of the post-quantum variant `pq-obfs` [GSV24] of Tor's `obfs4` protocol that enables hybrid security guarantees. We recall the purpose of an obfuscated key exchange protocol [GSV24] is to permit a client with prior knowledge of a server's public key to establish a fresh session key with the server, such that (1) no passive adversary can distinguish the handshake from a random sequence of messages of the same length and ordering; and (2) the server is otherwise quiet, meaning it does not respond to any messages sent by an adversary that does not demonstrate knowledge of the server's public key. The intention of the `obfs4` protocol in particular is to provide a layer of obfuscated traffic that does not match a censor's protocol blocklist and is secure against probing by censors. We state the security goals in more detail below.

1. **Key indistinguishability.** A Bellare–Rogaway-style [BR94] notion of key indistinguishability requires that the shared key derived in the protocol must be indistinguishable from random. Formally, an adversary actively interacts with multiple sessions, is allowed to reveal user and session keys, and must guess the challenge bit b used in a TEST oracle which returns either real or random session keys depending on b. (A Fresh predicate prevents trivial attacks like testing and revealing the same key.) We also aim for *forward secrecy*: session keys must remain secure if the involved server's long-term secret key is later compromised.

2. **Obfuscation.** The protocol transcript should be indistinguishable from a simulated transcript, where the *simulator* $\mathcal{S}$ is a parameter of the security definition. As with `pq-obfs`, we prove obfuscation with respect to a simulator that outputs random messages of varying lengths. This captures the idea that the protocol should lie within some *class* of protocols where all transmitted data appears to be random bits.

   An adversary's goal is to guess the challenge bit b used in a CHALLEXEC oracle which returns either real or simulated transcripts of the protocol (and real or random keys, respectively). The challenge bit b is the same as is used in TEST queries. The adversary must not violate the ObfFresh predicate, which requires that the server's secret key is not revealed (i.e., we aim for *strong obfuscation* as defined in [GSV24]).

3. **Probing resistance.** The protocol should be resistant to *active probing*, where censors probe suspicious proxy servers in an attempt to identify them. In particular, a responder should not respond to messages from an initiator who has not proven knowledge of the responder's public key. This is captured within the SEND oracle

and the Probed flag: If the adversary successfully elicits a non-empty response from a responder, whose public key is not revealed, in response to a message not previously sent by an honest initiator, then the adversary wins.

4. **Explicit authentication.** Finally, $\mathsf{ExplicitAuth}$ ensures explicit authentication of the server. An adversary wins if it causes a client to accept a session without a partnered server session existing.

4.1 Shortcomings of the `pq-obfs` Protocol

We identify a key issue that prevents `pq-obfs` from achieving hybrid guarantees, even when instantiated with a hybrid OKEM. In the first round of `pq-obfs`, the client, who knows the server's static KEM public key pk_S, encapsulates to pk_S, yielding (c_S, K_S). The client also generates an ephemeral keypair $(sk_e, pk_e) \leftarrow\!\!\$\ \mathsf{KEM.KGen}()$. In the first message, the client sends c_S and pk_e to the server. The server will decapsulate c_S and use pk_e to establish a fresh session key with the client. The issue with this construction is that pk_e is sent to the server *in the clear*, requiring public key uniformity ($\mathsf{pk\text{-}unif}$) from the ephemeral OKEM. However, we do not know of an existing OKEM with *hybrid* public key uniformity guarantees. In other words, `pq-obfs` with a hybrid OKEM has obfuscation that is only as strong as the weakest $\mathsf{pk\text{-}unif}$ of its underlying OKEMs, voiding the hybrid guarantees on the key exchange level.

4.2 The `Drivel` Protocol

We propose the `Drivel` protocol which addresses the above hybrid obfuscation barrier of `pq-obfs` while achieving the same security goals. In short, the key insight is to encrypt the ephemeral public key and ciphertext using a key derived from the shared secret from the static (O)KEM encapsulation. That way, `Drivel`, unlike `pq-obfs`, does not require public-key uniformity of the deployed (hybrid) ephemeral KEM, which is the only OKEM property for which OEINC does not achieve hybrid guarantees. Furthermore, additionally encrypting the ephemeral ciphertext means that for the ephemeral KEM, a regular (non-obfuscated) KEM suffices, improving efficiency. Instantiating `Drivel` with $\mathsf{OEINC}[\mathsf{DHKEM\text{-}Ell2}, \mathsf{ML\text{-}Kemeleon}, \oplus, \mathsf{G}, \mathsf{W}]$ (for reasonable PRG G and PRF W) as the static KEM and any hybrid $\mathsf{IND\text{-}1CCA}$-secure (non-obfuscated) KEM as the ephemeral KEM hence yields a readily implementable hybrid obfuscated key exchange protocol. The final structure remains closely aligned with `pq-obfs`, and, thus, the currently deployed `obfs4` protocol.

We describe the protocol at a high level. We retain from `pq-obfs` the general structure of ephemeral and static KEM encapsulations, and the key schedule (modulo deriving the additional encryption keys), so that we need only rely on standard assumptions (avoiding any random oracles). The full description can be found in Figure 6. In this setting, we assume that the server has published semi-private information, namely its OKEM public key and an identifier $NodeID \in \{0,1\}^{\mathsf{nl}}$, to a distribution service that only honest clients are intended to have access to. In practice, this can be a *bridge distribution service*, as used with Tor, or another out-of-band sharing mechanism. Although pre-shared semi-private information is a seemingly strong assumption, it is already an assumption of `obfs4`.

Client to server message. A client who knows the server's public key and $NodeID$ begins by generating a fresh ephemeral KEM keypair. It then establishes a shared secret with the server by encapsulating to the server's static OKEM public key. The client sends this ciphertext along with the ephemeral public key (encrypted, using a key derived from the static OKEM encapsulation), randomized padding, and MACs which include the server's public key and *NodeID*.

Server to client message. The server decapsulates and checks the MACs. On success, it encapsulates to the client's ephemeral public key, and sends back the ciphertext (encrypted using a key derived from the static OKEM encapsulation), along with padding and MACs. The user verifies these and derives the session key using the ephemeral-ephemeral and ephemeral-static KEM shared secrets, as well as the corresponding public keys and ciphertexts.

MACs. The MAC tags M_C and M_S are intended to aid in parsing the variable-length messages. In particular, the server may continue sending data immediately following its message to the client, and so the M_S marker identifies where the initial handshake message (and variable length padding) ends. The client's tag M_C is not strictly necessary, as the server may parse the handshake message from the end; however, we leave it in the description for symmetry and to allow for potentially more flexible designs in the future (e.g., the client sending data immediately following its handshake message).

Replay protection. We assume that, for replay protection, the protocol works in epochs; the client includes its epoch in MAC tag MAC_C. The server checks that this epoch is within a valid range, and stores the set of MAC values seen in its state $\mathcal{S}_{MAC}$ *per epoch.* To simplify the presentation, we omit epochs here.

Instantiating the protocol. Like `pq-obfs`, `Drivel` relies on a secure pseudorandom function F_1 with (default) output length fl_1 and a dual-PRF F_2 with output length fl_2. In addition, we employ an $\mathsf{OT\text{-}IND\$}$-secure symmetric encryption scheme SE. To handle potentially varying key lengths, we ask that F_1 also supports variable output lengths, where the output length is optionally specified through the function's third input (by default, it is fl_1). Unlike `pq-obfs`, `Drivel` does not require the ephemeral KEM to have public key or ciphertext uniformity. Instead, the client's ephemeral KEM public key is encrypted with a key derived from the static KEM, as is the server's ciphertext response. Thus, any $\mathsf{IND\text{-}1CCA}$-secure KEM can be used for the ephemeral KEM, and any OKEM with $\mathsf{IND\text{-}CCA}$ security, $\mathsf{SPR\text{-}CCA}$ security, and ciphertext uniformity can be used for the static OKEM.

This can be achieved, for example, with an obfuscated KEM constructed using OEINC from Section 3 (see Table 1 for examples). The ephemeral KEM can be instantiated with any regular (i.e., not necessarily obfuscated) KEM with $\mathsf{IND\text{-}1CCA}$ security (e.g., $\mathsf{X\text{-}Wing}$ [BCD$^+$24] would provide hybrid guarantees[4]). We can instantiate F_1 with $\mathsf{HKDF\text{-}Expand}$ [Kra10, KE10] and F_2 with HMAC [BCK96, KBC97]; the latter's dual-PRF security for fixed-length keys is proven in [BBGS23]. The symmetric encryption scheme, requiring only $\mathsf{OT\text{-}IND\$}$ security, can be instantiated with XOR with appropriate key lengths kl_1, kl_2 matching the ephemeral KEM's public key and ciphertext lengths.

4.3 Security

We use the security model of obfuscated key exchange from [GSV24], denoted sObfKE, discussed informally at the start of this section; we provide the full details in Appendix E for reference. In addition to the details of the model specific to the desired security properties

[4]In fact, even faster options are possible. Most lattice-based $\mathsf{IND\text{-}CCA}$ KEMs use a variant of the Fujisaki–Okamoto transform [FO99], which requires the decapsulator to re-encrypt the message to check for tampering. With the relaxation to $\mathsf{IND\text{-}1CCA}$, this step is no longer necessary [JMZ23].

```
                                                Server key generation/setup
                                                $NodeID \leftarrow_\$ \{0,1\}^{\mathsf{nl}}$
                                                $(pk_S, sk_S, \_) \leftarrow_\$ \mathsf{OKEM.KGen}()$
                                                $st.\mathcal{S}_{MAC} \leftarrow \emptyset$
                                                return $((sk_S, NodeID), (pk_S, NodeID), st)$

Client  knows $(pk_S, NodeID)$                                  knows $(sk_S, NodeID)$  Server
$(sk_e, pk_e) \leftarrow_\$ \mathsf{KEM.KGen}()$
$P_C \leftarrow_\$ \mathcal{D}_{\mathsf{pad_C}}$
$(c_S, K_S) \leftarrow_\$ \mathsf{OKEM.Encap}(pk_S)$
$ES \leftarrow \mathsf{F}_2(NodeID, K_S)$
$EK_1 \leftarrow \mathsf{F}_1(ES, \text{“}\texttt{:enckey1}\text{”}, \mathsf{kl}_1)$
$EK_2 \leftarrow \mathsf{F}_1(ES, \text{“}\texttt{:enckey2}\text{”}, \mathsf{kl}_2)$
$epk_e \leftarrow \mathsf{SE.Enc}(EK_1, pk_e)$
$M_C \leftarrow \mathsf{F}_1(ES, epk_e \| c_S \| \text{“}\texttt{:mc}\text{”})$
$MAC_C \leftarrow \mathsf{F}_1(ES, epk_e \| c_S \| P_C \| M_C \| \text{“}\texttt{:mac\_c}\text{”})$

                    $\xrightarrow{msg_C \,=\, epk_e \| c_S \| P_C \| M_C \| MAC_C}$

        $epk_e \leftarrow msg_C[1..\mathsf{ol}]$ ; $c_S \leftarrow msg_C[\mathsf{ol}+1..\mathsf{ol}+\mathsf{cl}]$
        $K_S \leftarrow \mathsf{OKEM.Decap}(sk_S, c_S)$
        $ES \leftarrow \mathsf{F}_2(NodeID, K_S)$
        $EK_1 \leftarrow \mathsf{F}_1(ES, \text{“}\texttt{:enckey1}\text{”}, \mathsf{kl}_1)$
        $EK_2 \leftarrow \mathsf{F}_1(ES, \text{“}\texttt{:enckey2}\text{”}, \mathsf{kl}_2)$
        $M_C \leftarrow \mathsf{F}_1(ES, epk_e \| c_S \| \text{“}\texttt{:mc}\text{”})$
        parse $(epk_e \| c_S \| P_C \| M_C \| MAC_C) \leftarrow msg_C$ using $M_C$; else break
        if $\mathsf{F}_1(ES, epk_e \| c_S \| P_C \| M_C \| \text{“}\texttt{:mac\_c}\text{”}) \neq MAC_C$: break
        if $MAC_C \in st.\mathcal{S}_{MAC}$: break
        $pk_e \leftarrow \mathsf{SE.Dec}(EK_1, epk_e)$
        $(c_e, K_e) \leftarrow_\$ \mathsf{KEM.Encap}(pk_e)$
        $ect_e \leftarrow \mathsf{SE.Enc}(EK_2, c_e)$
        $protoID \leftarrow \text{“}\texttt{Drivel}\text{”}$
        $ES' \leftarrow \mathsf{F}_1(ES, \text{“}\texttt{:derive\_key}\text{”})$ ; $FS \leftarrow \mathsf{F}_2(ES', K_e)$
        $context \leftarrow pk_S \| c_S \| pk_e \| c_e \| protoID$
        $skey \leftarrow \mathsf{F}_1(FS, context \| \text{“}\texttt{:key\_extract}\text{”})$
        $auth \leftarrow \mathsf{F}_1(FS, context \| \text{“}\texttt{:server\_mac}\text{”})$
        $P_S \leftarrow_\$ \mathcal{D}_{\mathsf{pad_S}}$
        $M_S \leftarrow \mathsf{F}_1(ES, ect_e \| \text{“}\texttt{:ms}\text{”})$
        $MAC_S \leftarrow \mathsf{F}_1(ES, ect_e \| auth \| P_S \| M_S \| \text{“}\texttt{:mac\_s}\text{”})$

                    $\xleftarrow{msg_S \,=\, ect_e \| auth \| P_S \| M_S \| MAC_S}$

$ect_e \leftarrow msg_S[1..\mathsf{cl}]$
$M_S \leftarrow \mathsf{F}_1(ES, ect_e \| \text{“}\texttt{:ms}\text{”})$
parse $(ect_e \| auth \| P_S \| M_S \| MAC_S) \leftarrow msg_S$ using $M_S$; else break
if $\mathsf{F}_1(ES, ect_e \| auth \| P_S \| M_S \| \text{“}\texttt{:mac\_s}\text{”}) \neq MAC_S$: break
$c_e \leftarrow \mathsf{SE.Dec}(EK_2, ect_e)$
$K_e \leftarrow \mathsf{KEM.Decap}(sk_e, c_e)$
$protoID \leftarrow \text{“}\texttt{Drivel}\text{”}$
$ES' \leftarrow \mathsf{F}_1(ES, \text{“}\texttt{:derive\_key}\text{”})$ ; $FS \leftarrow \mathsf{F}_2(ES', K_e)$
$context \leftarrow pk_S \| c_S \| pk_e \| c_e \| protoID$
$skey \leftarrow \mathsf{F}_1(FS, context \| \text{“}\texttt{:key\_extract}\text{”})$
if $\mathsf{F}_1(FS, context \| \text{“}\texttt{:server\_mac}\text{”}) \neq auth$: break
```

Figure 6: The `Drivel` obfuscated key exchange protocol. OKEM is an OKEM satisfying IND-CCA, SPR-CCA, and ciphertext uniformity. KEM is an IND-1CCA-secure KEM. SE is an OT-IND\$-secure symmetric encryption scheme. F_1 is a PRF and F_2 is a dual PRF. Core differences to the `pq-obfs` protocol from [GSV24] are highlighted in blue boxes.

described above, session identifiers are used to determine that two sessions are partnered, while contributive identifiers are used to determine when a responder session has an honest communication partner. The security analysis makes use of a selective security variant of the game, where the adversary has to commit upfront to winning via (1) a single Test query to a pre-declared session, or (2) a single ChallExec query against a pre-declared server. This variant implies full security via a hybrid argument, as shown in [GSV24].

Simulator definition. We establish sObfKE security with respect to the following simulator $\mathcal{S}_{\texttt{Drivel}}$ which outputs two uniformly random messages of length dependent on the `Drivel` distribution of padding:

```
$\underline{\mathcal{S}_{\texttt{Drivel}}}$:
$P_C \leftarrow\$ \mathcal{D}_{\mathsf{pad_C}}$ ; $P_S \leftarrow\$ \mathcal{D}_{\mathsf{pad_S}}$ // sample client/server padding according to distribution
$m_1 \leftarrow\$ \{0,1\}^{\mathsf{epl}+\mathsf{cl}+2\cdot\mathsf{fl}_1+|P_C|}$ // 1x encrypted pk + 1x ctxt + 2x $\mathsf{F}_1$ outputs (MACs) + client padding
$m_2 \leftarrow\$ \{0,1\}^{\mathsf{ecl}+3\cdot\mathsf{fl}_1+|P_S|}$ // 1x encrypted ctxt + 3x $\mathsf{F}_1$ outputs (auth and MACs) + server padding
return $(m_1, m_2)$
```

Here, $\mathsf{epl}, \mathsf{ecl}$ are the lengths of the encrypted ephemeral KEM public key and ciphertext, respectively, fl_1 is the (default) output length of the PRF F_1, and cl is the ciphertext length of OKEM.

Session and contributive identifiers. We set the session identifier as $sid := (pk_e, c_e, pk_S, c_S)$ where pk_e is the initiator's KEM public key and c_e is the corresponding KEM ciphertext, and pk_S is the responder's static KEM public key and c_S the corresponding ciphertext. We set the contributive identifier to $cid := (pk_e, c_S)$ upon the client sending or server receiving the first message.

Theorem 4.1. *Let* `Drivel` *be defined as in Figure 6. For any* sObfKE *adversary* $\mathcal{A}$ *against* `Drivel`*, we give algorithms* $\mathcal{B}_1$*–*$\mathcal{B}_{17}$ *such that*

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{sObfKE}}_{\texttt{Drivel},\mathcal{S}_{\texttt{Drivel}}}(\mathcal{A}) \leq 2 \cdot \Bigg(& \mathsf{pkcoll}_{\mathsf{KEM}}(n_s) + \mathsf{pkcoll}_{\mathsf{OKEM}}(n_r) + n_s \cdot (\delta_{\mathsf{OKEM}} + \delta_{\mathsf{KEM}}) \\
& + n_s n_r \cdot \Big(\mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_2}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_1}(\mathcal{B}_2) + \frac{1}{2^{\mathsf{fl}_1}} \Big) \\
& + n_s n_r \cdot \Big(\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{OKEM}}(\mathcal{B}_3) + \mathsf{Adv}^{\mathsf{swap\text{-}PRF}}_{\mathsf{F}_2}(\mathcal{B}_4) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_1}(\mathcal{B}_5) \\
& \qquad + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_2}(\mathcal{B}_6) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_1}(\mathcal{B}_7) + \frac{1}{2^{\mathsf{fl}_1}} \Big) \\
& + (n_s + n_r q_{\mathrm{C}}) \cdot \Big(n_s \cdot \Big(\mathsf{Adv}^{\mathsf{IND\text{-}1CCA}}_{\mathsf{KEM}}(\mathcal{B}_8) + \mathsf{Adv}^{\mathsf{swap\text{-}PRF}}_{\mathsf{F}_2}(\mathcal{B}_9) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_1}(\mathcal{B}_{10}) \Big) \\
& \qquad + \mathsf{Adv}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{OKEM},\mathcal{S}_\$}(\mathcal{B}_{11}) + \mathsf{Adv}^{\mathsf{swap\text{-}PRF}}_{\mathsf{F}_2}(\mathcal{B}_{12}) \\
& \qquad + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_1}(\mathcal{B}_{13}) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_2}(\mathcal{B}_{14}) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_1}(\mathcal{B}_{15}) \\
& \qquad + \mathsf{Adv}^{\mathsf{OT\text{-}IND\$}}_{\mathsf{SE}}(\mathcal{B}_{16}) + \mathsf{Adv}^{\mathsf{OT\text{-}IND\$}}_{\mathsf{SE}}(\mathcal{B}_{17}) \Big) \Bigg),
\end{aligned}$$

where $\mathcal{A}$ *makes at most* q_{C} ChallExec *queries;* n_s, n_r *are the number of sessions and servers (parties in* responder *role) that* $\mathcal{A}$ *interacts with, respectively;* fl_1 *is the output bit-length of* F_1*; and* $\mathcal{S}_\$$ *outputs a random ciphertext from* $\{0,1\}^{\mathsf{cl}}$.

Due to the intentional similarity of `Drivel` to `pq-obfs`, the proof of Theorem 4.1 follows closely (in large parts verbatim) the security proof for `pq-obfs` from [GSV24]. Here, we provide a proof sketch with the <u>main changes compared to the proof for `pq-obfs` underlined</u>; the complete proof is deferred to Appendix F.

Proof sketch. We proceed via a series of game hops and branches, considering each clause in FINALIZE. We rule out KEM public key collisions (term $\mathsf{pkcoll}_{\mathsf{KEM}}(n_s) + \mathsf{pkcoll}_{\mathsf{OKEM}}(n_r)$) and assume correct decapsulation in all sessions ($n_s \cdot (\delta_{\mathsf{OKEM}} + \delta_{\mathsf{KEM}})$), which ensures soundness (Sound).

A first branch then rules out that $\mathcal{A}$ successfully probes a server (Probed): we guess the first server and session that sets Probed (with a $n_s \cdot n_r$ loss). From that server's uncompromised semi-private *NodeID*, we apply PRF security twice to turn ES and then MAC_C into random values. After this, we observe that Probed is set if a non–initiator-first message m yields a reply by an unrevealed responder. Such a message m is either rejected due to the replay check, or it is different from all honest initiators' first messages sent to this responder. In the second case, m contains the target client MAC value MAC_C^* which is a random fl_1-bit value unknown to $\mathcal{A}$. The adversary $\mathcal{A}$ can therefore only guess MAC_C^*; the corresponding bound is $\mathsf{Adv}_{\mathsf{F}_2}^{\mathsf{PRF}}(\mathcal{B}_1) + \mathsf{Adv}_{\mathsf{F}_1}^{\mathsf{PRF}}(\mathcal{B}_2) + 1/2^{\mathsf{fl}_1}$.

A second branch prevents explicit authentication ($\mathsf{ExplicitAuth}$) from being violated. We guess the first session violating $\mathsf{ExplicitAuth}$, π^*, and its peer, v^*, (losing a factor $n_s \cdot n_r$). Then we can embed an $\mathsf{IND\text{-}CCA}$ challenge in that the peer's public key, the session's c_S, and the derived K_S shared secret (term $\mathsf{Adv}_{\mathsf{OKEM}}^{\mathsf{IND\text{-}CCA}}(\mathcal{B}_3)$). Let $(c_S, K_S) \leftarrow\!\!\$\ \mathsf{OKEM.Encap}(pk_S)$ be the encapsulation computed in the target session π^* with the long-term public key of v^*. We replace the long-term shared secret K_S with a uniformly random $\widetilde{K_S}$ in π^*. All values derived from K_S in π^* use the randomized value $\widetilde{K_S}$. We bound the adversary $\mathcal{A}$'s difference in advantage by a reduction $\mathcal{B}_3$ to the $\mathsf{IND\text{-}CCA}$ security of OKEM. Next, through four (swap-)PRF hops ($\mathcal{B}_4$–$\mathcal{B}_7$), we show that this turns *auth* into a random fl_1-bit value leaving $\mathcal{A}$ with a $1/2^{\mathsf{fl}_1}$ chance to violate $\mathsf{ExplicitAuth}$.

We then restrict the adversary to a single-challenge ($\mathsf{sObfKE\text{-}1}$) version of the game, applying the hybrid argument from [GSV24, Theorem 4.3] with a $(n_s + n_r q_{\mathrm{C}})$ loss. We separately treat the following two cases.

For Case I ($\mathcal{A}$ makes a single TEST query), we first guess the (*sid*- or *cid*-) partner session of the tested session (losing a factor n_s). We then embed an $\mathsf{IND\text{-}1CCA}$ challenge in the ephemeral KEM encapsulation of the test session, π^*, and its partner session, π_p^* (term $\mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{IND\text{-}1CCA}}(\mathcal{B}_8)$). In particular, we replace the ephemeral KEM key K_e with a uniformly random $\widetilde{K_e}$ in the target session π^*. All values derived from K_e in π^* use the randomized value $\widetilde{K_e}$. We bound the adversary $\mathcal{A}$'s difference in advantage by a reduction $\mathcal{B}_8$ to the $\mathsf{IND\text{-}1CCA}$ security of OKEM. Following this, we apply two (swap-)PRF hops ($\mathcal{B}_9$, $\mathcal{B}_{10}$) to randomize the test session's key, concluding this case.

For Case II ($\mathcal{A}$ makes a single CHALLEXEC query to a pre-determined server), we embed a $\mathsf{SPR\text{-}CCA}$ challenge in the long-term KEM encapsulation of the CHALLEXEC sessions (term $\mathsf{Adv}_{\mathsf{OKEM}}^{\mathsf{SPR\text{-}CCA}}(\mathcal{B}_{11})$), randomizing both c_S and K_S. Specifically, we replace the ciphertext c_S, as well as the long-term KEM key K_S as follows. Instead of running $\mathsf{OKEM.Encap}$, the initiator samples $c_S \leftarrow\!\!\$\ \{0,1\}^{\mathsf{cl}}$ and $K_S \leftarrow\!\!\$\ \mathcal{K}$ uniformly at random. We bound the adversary $\mathcal{A}$'s difference in advantage by a reduction $\mathcal{B}_{11}$ to the $\mathsf{SPR\text{-}CCA}$ security of OKEM. Subsequently, we proceed via four (swap-)PRF hops ($\mathcal{B}_{12}$–$\mathcal{B}_{15}$) to randomize the transcript's MAC values as well as the encryption keys and session key. <u>The main difference from the proof of `pq-obfs` is that in the final steps, we apply the $\mathsf{OT\text{-}IND\$}$ security of the</u>

symmetric encryption scheme SE to randomize epk_e and ect_e, corresponding to the terms $\mathsf{Adv}_{\mathsf{SE}}^{\mathsf{OT\text{-}IND\$}}(\mathcal{B}_{16})$ and $\mathsf{Adv}_{\mathsf{SE}}^{\mathsf{OT\text{-}IND\$}}(\mathcal{B}_{17})$. At this point, the protocol messages in ChallExec sessions are uniformly distributed like outputs of the simulator $\mathcal{S}_{\texttt{Drivel}}$, concluding this case and the proof. $\square$

4.4 Additional Features

We now discuss optional features of `Drivel` which are intended to aid in development and deployment, but are not included in our security analysis.

Strong obfuscation vs. DoS resistance. `Drivel`, as described thus far, has *strong obfuscation*, meaning that even an adversary who learns the server's static public key pk_s (recall that this is considered semi-private information in our setting) is not able to passively distinguish a protocol's transcript from a simulated one. The cost of this advantage is that the server must perform a decapsulation for every incoming message, even when the other party does not know the server's *NodeID*. Since decapsulations can be costly, this may be used by an attacker to launch a denial-of-service attack against the server.[5]

In order to accommodate different deployment environments, the client can optionally send a *cookie* in the first round of communication, as was done in the original `obfs4` protocol as well as Wireguard [Don17]. Concretely, this cookie may be computed as $\mathsf{HMAC}(pk_s \| NodeID, epk_e \| c_S \| P_C \| M_C \| MAC_C)$ (the difference from MAC_C being that a static key $pk_s \| NodeID$ is used rather than an ephemeral key ES). The server can then check this cookie prior to performing decapsulation, preventing the processing of messages from clients that do not know *NodeID*. To prevent replays, the server must store past cookies.

Finally, we note that there are middle ground options between strong and regular obfuscation. For example, a server may publish a temporary pseudorandom token *tok*, along with its public key and *NodeID*, and the client may compute the cookie as $\mathsf{HMAC}(tok \| pk_s \| NodeID, epk_e \| c_S \| P_C \| M_C \| MAC_C)$. Thus, if a future token is ever revealed, past protocols using old tokens are still not distinguishable from random. Temporary tokens can be eschewed entirely if the server periodically updates its public key pk_s in a way that is not publicly reversible (e.g., by generating a brand new key or re-randomizing an existing key using secret randomness). However for lattice-based OKEMs, public keys are far larger than strictly necessary to achieve this strong(er) obfuscation property.

Padding distribution. Another difference to `pq-obfs` is that we enable an arbitrary distribution of padding to be used. Previously, the padding length (and consequently, the message length) was chosen uniformly at random from a fixed range. An arbitrary padding distribution provides more flexibility in traffic patterns, allowing, for example, a padding distribution that fixes message lengths to match the traffic pattern of another protocol.

Extra data. One may optionally place data encrypted under a key derived from K_S in place of (or in addition to) the padding of the initial message to the server. This is analogous to the *extra data* feature of `ntor` version 3 [Mat21]. We note that this extra data does not benefit from forward secrecy because it is encrypted under a key derived from the static KEM encapsulation alone.

[5]This tradeoff between strong obfuscation and DoS resistance in `obfs4` has been previously noted [Ang22].

$\underline{\mathbf{A}(sid, pw)}$		$\underline{\mathbf{B}(sid, pw)}$
$(pk, sk) \leftarrow_\$ \mathsf{KGen}()$		
$epk \leftarrow \mathsf{E}_{pw}^{sid\|1}(pk)$	$\xrightarrow{epk}$	$pk \leftarrow \mathsf{D}_{pw}^{sid\|1}(epk)$
		$(c, Z) \leftarrow_\$ \mathsf{Encap}(pk)$
$c \leftarrow \mathsf{D}_{pw}^{sid\|2}(ec)$	$\xleftarrow{ec}$	$ec \leftarrow \mathsf{E}_{pw}^{sid\|2}(c)$
$Z \leftarrow \mathsf{Decap}_{sk}(c)$		
$K \leftarrow \mathsf{H}(sid, A, B, epk, ec, Z)$		$K \leftarrow \mathsf{H}(sid, A, B, epk, ec, Z)$
return K		**return** K

Figure 7: The CAKE PAKE [BCP+23]. (E, D) are the encryption and decryption algorithms of an ideal cipher. *sid* is the protocol execution's session identifier, and A, B are the party identifiers.

5 Hybrid PAKE with Adaptive Security

In this section we describe a method for constructing a password-authenticated key exchange (PAKE) from an OKEM. When the OKEM has hybrid guarantees, then so does the resulting PAKE. We distinguish our approach from previous constructions [HR24, LL24] in two ways. First, our approach achieves adaptive, rather than static, security, since the underlying PAKE protocol is proven to realize the $\mathcal{F}_{\mathsf{PAKE}}$ ideal functionality in the Universal Composability model [Can01] with adaptive corruptions. This answers the open question in [HR24] of whether such a scheme exists. Second, our approach achieves a new tradeoff between round complexity and runtime: previous hybrid PAKE constructions include efficient three-round protocols which use standard lattice-based KEMs, as well as more computationally intensive one-round protocols using isogenies; our protocol has two rounds, and has computational overhead that lies in between the two.

5.1 CAKE

We first provide an overview of CAKE [BCP+23], an adaptively-secure PAKE that is generically constructed from an underlying KEM with certain properties. We observe that an OKEM with ciphertext and public key uniformity is sufficient to instantiate CAKE. Then, we construct a hybrid OKEM with unconditional public key uniformity. Together, this yields the first hybrid adaptively-secure PAKE.

Protocol. A graphical overview of the CAKE protocol can be found in Figure 7. The protocol begins with the initiator generating a fresh KEM keypair, encrypting the public key using an ideal cipher with *pw* as the key, and sending the result to the responder. The responder decrypts the public key, encapsulates to it, and encrypts the encapsulation using the ideal cipher with the same password. These ideal ciphers are domain-separated, represented in the diagram by two distinct session IDs in the superscript of the algorithm. Finally, the initiator decrypts and decapsulates. The session key is the hash of the KEM shared secret along with the protocol transcript.

Required KEM properties. CAKE requires its underlying KEM to have IND-CPA security, anonymity (Definition B.2), and fuzziness (Definition B.1). Recall that when public keys and ciphertexts are bitstrings, anonymity is strictly weaker than ciphertext uniformity and public key uniformity (Definition 2.6) is equivalent to fuzziness (these claims are treated more rigorously in Appendix B). Thus, CAKE can be instantiated with an OKEM with IND-CPA security, ciphertext uniformity, and public key uniformity. Further, since encoded public keys are uniform-looking bitstrings, we may instantiate the ideal cipher with a tweakable wide block cipher, and use the tweak for domain separation. This is easier to implement and more efficient than, for example, a custom, constant-time, keyed permutation on the range $[0, q^{768})$, as suggested by the CAKE authors for ML-KEM-768 public keys.

5.2 Achieving Hybrid CAKE

Using the OKEM combiner of Section 3, applied to, e.g., DHKEM-Ell2 (outer) and ML-Kemeleon (inner), we obtain an OKEM with hybrid IND-CCA (and, hence, IND-CPA) security and ciphertext uniformity, but not public key uniformity. Recall that the public key uniformity of the combined OKEM is the *weakest* of the underlying public key uniformity properties. Thus, to construct CAKE with hybrid guarantees, we will need to combine two OKEMs with statistical public key uniformity. Since our outer KEM already has this property, it remains only to construct a post-quantum OKEM with statistical public key uniformity. No post-quantum OKEM we have mentioned so far achieves this; however, we are able to construct one based on unstructured LWE. We follow the technique hinted at in [HR24] for constructing a one-round lattice-based PAKE from (superpolynomial modulus) LWE.

Post-quantum statistical public key uniformity. As a starting point, we consider FrodoKEM [BCD+16], an LWE-based KEM and also a natural OKEM (its public keys and ciphertexts already perfectly pack into bitstrings). It will suffice to use its IND-CPA-secure variant, also called FrodoPKE. FrodoKEM is a *double*-LWE KEM: its public keys and ciphertexts are both LWE samples. This technique, first described in [LP11], affords the scheme a meaningful improvement in both security guarantees and ciphertext size compared to a Regev-style scheme [Reg05], wherein only one of the aforementioned values is an LWE sample. Unfortunately, double-LWE schemes enjoy only computational public key uniformity, as that uniformity relies on the hardness of LWE.

We construct a KEM with statistical public key uniformity by simply undoing the optimization from [LP11]. Concretely, rather than defining the public key vector as an LWE sample $\mathbf{b} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}$, we define it as the matrix product $\mathbf{b} \leftarrow \mathbf{A}\mathbf{s}$. To retain security, we must modify the dimensions of $\mathbf{A}$ and $\mathbf{s}$. Rather than using a square matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$, we use a rectangular $\mathbf{A} \in \mathbb{Z}_q^{n \times n'}$ where $n' \geq n \log_2 q$. This is necessary to make public keys statistically close to uniform via the leftover hash lemma [HILL99, LP11]. We call this construction, after appropriate lifting to the full dimension of the public key and ciphertext, StatFrodoKEM.

Performance tradeoffs. Due to the necessary parameter changes, ciphertexts grow by a factor of about $\log_2 q$. For StatFrodoKEM, this would result in 144kB ciphertexts in the level I parameter set, up from 9.6kB.[6] This would also cause encapsulation and key generation time to increase by a similar factor.

CAKE with hybrid OKEM offers a new set of tradeoffs compared to existing hybrid PAKEs. We summarize these tradeoffs in Table 2. This construction is two rounds, rather than the three-round sequential combiners and one-round parallel combiners of [HR24, LL24]. In addition, it requires less overall computation than the one-round parallel combiner of

[6]There may be routes to improve ciphertext size using ideal-lattice-based schemes and leftover hash analogues [LPR13, DSGKS20], but we leave this to future work.

Table 2: Comparison of different hybrid PAKE mechanisms at the 128-bit security level. Estimates of runtime are in cycles on an Intel Skylake CPU, and exclude communication time. We also include communication overhead and indicate whether a security proof exists in the adaptive corruption setting. ParComb and SeqComb are the parallel and sequential combiners defined in [HR24] and identically in [LL24]. X-GA-PAKE is assumed to use the CTIDH group action [BBC+21] and passwords of bitlength $\ell = 80$.

Scheme	Rounds	Runtime (cycles)	Comm. (kB)	Adaptive corruptions
ParComb[X-GA-PAKE, SPAKE2]	1	5.0×10^{10}	20.5	x
CAKE[OEINC[DHKEM-Ell2, StatFrodoKEM]]	2	1.3×10^{8}	153.8	✓
SeqComb[CAKE[FrodoKEM], SPAKE2]	3	2.3×10^{7}	19.4	x

[LL24], which relies on X-GA-PAKE, a supersingular isogeny group action PAKE [AEK+22] (as well as on a classical PAKE such as SPAKE2 [AP05]). Inferring from benchmarks on Intel Skylake CPUs [BBC+21, BCD+16] and assuming linear scaling for matrix multiplication, we estimate the full runtime of CAKE (ignoring communication costs) to be 384× faster than the one-round alternative. This comes at the cost of an extra communication round and communication overhead of 7.5×.

Other hybrid PAKEs. A similar set of tradeoffs, albeit without security against adaptive corruptions, can be achieved by instantiating other KEM-based PAKEs such as OCAKE [BCP+23] and CHIC [ABJS24][7] with our hybrid OKEM.

6 Future Work

We briefly describe some unresolved questions, which we leave to future work.

UDP-compatible obfuscated key exchange. A common payload limit for UDP packets on the Internet is 1472 bytes. This is not an issue for Wireguard, which uses elliptic curve Diffie–Hellman for its key exchange. However, when instantiated with some post-quantum KEMs, the first round of `Drivel` exceeds this threshold. Since IPv4 fragmentation is frequently left disabled due to abuse concerns, there is no obvious way to perform key exchange over UDP while retaining obfuscation properties. Sequence numbers would violate obfuscation requirements, and using any pre-shared information to encrypt data (e.g., *NodeID*) would surrender strong obfuscation. Another challenge is managing retransmission as a result of fragmentation. In addition, stream IDs are necessary for users to be able to *roam*, i.e., maintain a connection while changing their IP or outgoing port. Unfortunately, embedding a stream ID would break obfuscation entirely, and using a symmetric key would either break strong obfuscation, or require the server to do trial decryptions over its set of session keys. It is thus an open problem whether these convenient properties can be recovered in fully encrypted protocols with post-quantum key exchange.

Hybrid ANO-CCA KEMs. Our combiner OEINC immediately yields a KEM providing

[7]CHIC requires the KEM to be *splittable*, i.e., that all public keys contain a uniform bitstring. ML-KEM is already splittable, since the public matrix seed is a uniform bitstring. In addition, DHKEMs are trivially splittable: we can simply add a sample from $\{0,1\}^\lambda$ to the end of any DHKEM public key. Since combining splittable OKEMs yields a splittable OKEM (via concatenation), we can instantiate hybrid CHIC.

hybrid anonymity (ANO-CCA), since SPR-CCA implies ANO-CCA [Xag22]. While SPR-CCA security provides a clear avenue for proving anonymity, some of the properties we required in our combiner (in particular, *strong* ciphertext uniformity) arise specifically to enable the proof of SPR-CCA. Thus, it is interesting to ask whether a hybrid ANO-CCA KEM (or anonymous KEM combiner) can be constructed with weaker assumptions on the two input KEMs, perhaps using a method other than SPR-CCA to prove anonymity.

Acknowledgements

D.S. is supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery grant RGPIN-2022-03187 and NSERC Alliance grant ALLRP 578463-22.

References

[ABH+21] Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. Analysing the HPKE standard. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 87–116. Springer, Cham, October 2021. doi:10.1007/978-3-030-77870-5_4.

[ABJS24] Afonso Arriaga, Manuel Barbosa, Stanislaw Jarecki, and Marjan Skrobot. C'est très CHIC: A compact password-authenticated key exchange from lattice-based KEM. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024*, pages 3–33, Singapore, 2024. Springer Nature Singapore.

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Berlin, Heidelberg, April 2001. doi:10.1007/3-540-45353-9_12.

[AEK+22] Michel Abdalla, Thorsten Eisenhofer, Eike Kiltz, Sabrina Kunzweiler, and Doreen Riepel. Password-authenticated key exchange from group actions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 699–728. Springer, Cham, August 2022. doi:10.1007/978-3-031-15979-4_24.

[And12] Nate Anderson. Stakeout: how the FBI tracked and busted a Chicago Anon, March 2012. https://arstechnica.com/tech-policy/2012/03/stakeout-how-the-fbi-tracked-and-busted-a-chicago-anon/.

[Ang22] Yawning Angel. Comment in Issue "Improve the obfs4 obfuscation protocol (#30716)" · The Tor Project / Anti-censorship / Pluggable Transports / lyrebird · GitLab, August 2022. URL: https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transports/lyrebird/-/issues/30716#note_2833105.

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 191–208. Springer, Berlin, Heidelberg, February 2005. doi:10.1007/978-3-540-30574-3_14.

[BBB$^+$24] Daniel J. Bernstein, Karthikeyan Bhargavan, Shivam Bhasin, Anupam Chattopadhyay, Tee Kiah Chia, Matthias J. Kannwischer, Franziskus Kiefer, Thales Paiva, Prasanna Ravi, and Goutam Tamvada. KyberSlash: Exploiting secret-dependent division timings in kyber implementations. Cryptology ePrint Archive, Report 2024/1049, 2024. URL: `https://eprint.iacr.org/2024/1049`.

[BBC$^+$21] Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. CTIDH: faster constant-time CSIDH. *IACR TCHES*, 2021(4):351–387, 2021. URL: `https://tches.iacr.org/index.php/TCHES/article/view/9069`, `doi:10.46586/tches.v2021.i4.351-387`.

[BBC$^+$22] John Baena, Pierre Briaud, Daniel Cabarcas, Ray A. Perlner, Daniel Smith-Tone, and Javier A. Verbel. Improving support-minors rank attacks: Applications to GeMSS and rainbow. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 376–405. Springer, Cham, August 2022. `doi:10.1007/978-3-031-15982-4_13`.

[BBDP01] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 566–582. Springer, Berlin, Heidelberg, December 2001. `doi:10.1007/3-540-45682-1_33`.

[BBF$^+$19] Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Goncalves, and Douglas Stebila. Hybrid key encapsulation mechanisms and authenticated key exchange. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*, pages 206–226. Springer, Cham, 2019. `doi:10.1007/978-3-030-25510-7_12`.

[BBGS23] Matilda Backendal, Mihir Bellare, Felix Günther, and Matteo Scarlata. When messages are keys: Is HMAC a dual-PRF? In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 661–693. Springer, Cham, August 2023. `doi:10.1007/978-3-031-38548-3_22`.

[BBLW22] R. Barnes, K. Bhargavan, B. Lipp, and C. Wood. Hybrid Public Key Encryption. RFC 9180 (Informational), February 2022. URL: `https://www.rfc-editor.org/rfc/rfc9180.txt`, `doi:10.17487/RFC9180`.

[BCD$^+$16] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1006–1018. ACM Press, October 2016. `doi:10.1145/2976749.2978425`.

[BCD$^+$24] Manuel Barbosa, Deirdre Connolly, João Diogo Duarte, Aaron Kaiser, Peter Schwabe, Karoline Varner, and Bas Westerbaan. X-wing. *CiC*, 1(1):21, 2024. `doi:10.62056/a3qj89n4e`.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 1–15. Springer, Berlin, Heidelberg, August 1996. doi:10.1007/3-540-68697-5_1.

[BCP$^+$23] Hugo Beguinet, Céline Chevalier, David Pointcheval, Thomas Ricosset, and Mélissa Rossi. GeT a CAKE: Generic transformations from key encaspulation mechanisms to password authenticated key exchanges. In Mehdi Tibouchi and Xiaofeng Wang, editors, *ACNS 23International Conference on Applied Cryptography and Network Security, Part II*, volume 13906 of *LNCS*, pages 516–538. Springer, Cham, June 2023. doi:10.1007/978-3-031-33491-7_19.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Berlin, Heidelberg, April 2006. doi:10.1007/11745853_14.

[Beu22] Ward Beullens. Breaking rainbow takes a weekend on a laptop. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 464–479. Springer, Cham, August 2022. doi:10.1007/978-3-031-15979-4_16.

[BFGJ17] Jacqueline Brendel, Marc Fischlin, Felix Günther, and Christian Janson. PRF-ODH: Relations, instantiations, and impossibility results. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 651–681. Springer, Cham, August 2017. doi:10.1007/978-3-319-63697-9_22.

[BHKL13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 967–980. ACM Press, November 2013. doi:10.1145/2508859.2516734.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Berlin, Heidelberg, May 2000. doi:10.1007/3-540-45539-6_11.

[BR94] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Berlin, Heidelberg, August 1994. doi:10.1007/3-540-48329-2_21.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. doi:10.1109/SFCS.2001.959888.

[CD23] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 423–447. Springer, Cham, April 2023. doi:10.1007/978-3-031-30589-4_15.

[DFGS15] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1197–1210. ACM Press, October 2015. doi:10.1145/2810103.2813653.

[DHRR22a] Benjamin Dowling, Eduard Hauck, Doreen Riepel, and Paul Rösler. Strongly anonymous ratcheted key exchange. Cryptology ePrint Archive, Report 2022/1187, 2022. URL: https://eprint.iacr.org/2022/1187.

[DHRR22b] Benjamin Dowling, Eduard Hauck, Doreen Riepel, and Paul Rösler. Strongly anonymous ratcheted key exchange. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 119–150. Springer, Cham, December 2022. doi:10.1007/978-3-031-22969-5_5.

[DKRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 18*, volume 10831 of *LNCS*, pages 282–305. Springer, Cham, May 2018. doi:10.1007/978-3-319-89339-6_16.

[Don17] Jason A. Donenfeld. WireGuard: Next generation kernel network tunnel. In *NDSS 2017*. The Internet Society, February / March 2017. doi:10.14722/ndss.2017.23160.

[DSGKS20] Dana Dachman-Soled, Huijing Gong, Mukul Kulkarni, and Aria Shahverdi. Towards a ring analogue of the leftover hash lemma. *Journal of Mathematical Cryptology*, 15:87–110, 2020. URL: https://api.semanticscholar.org/CorpusID:227129476.

[FJ24] Ellis Fenske and Aaron Johnson. Bytes to schlep? Use a FEP: Hiding protocol metadata with fully encrypted protocols. In *ACM CCS 2024*, CCS '24, page 1982–1996, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3658644.3690198.

[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, Berlin, Heidelberg, August 1999. doi:10.1007/3-540-48405-1_34.

[Fre22] French Cybersecurity Agency. ANSSI views on the post-quantum cryptography transition, March 2022. https://cyber.gouv.fr/sites/default/files/document/EN_Position.pdf.

[Ger24] German Federal Office for Information Security. BSI TR-02102-1, cryptographic mechanisms: Recommendations and key lengths, February 2024. Version 2024-01, https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf.

[GHJ+22] Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. Don't reject this: Key-recovery timing attacks due to rejection-sampling in HQC and BIKE. *IACR TCHES*, 2022(3):223–263, 2022. doi:10.46586/tches.v2022.i3.223-263.

[GHP18] Federico Giacon, Felix Heuer, and Bertram Poettering. KEM combiners. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 190–218. Springer, Cham, March 2018. `doi:10.1007/978-3-319-76578-5_7`.

[GJK21] Yanqi Gu, Stanislaw Jarecki, and Hugo Krawczyk. KHAPE: Asymmetric PAKE from key-hiding key exchange. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 701–730, Virtual Event, August 2021. Springer, Cham. `doi:10.1007/978-3-030-84259-8_24`.

[GJN20] Qian Guo, Thomas Johansson, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 359–386. Springer, Cham, August 2020. `doi:10.1007/978-3-030-56880-1_13`.

[GMP22] Paul Grubbs, Varun Maram, and Kenneth G. Paterson. Anonymous, robust post-quantum public key encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 402–432. Springer, Cham, May / June 2022. `doi:10.1007/978-3-031-07082-2_15`.

[GSU13] Ian Goldberg, Douglas Stebila, and Berkant Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *DCC*, 67(2):245–269, 2013. `doi:10.1007/s10623-011-9604-z`.

[GSV24] Felix Günther, Douglas Stebila, and Shannon Veitch. Obfuscated key exchange. In *ACM CCS 2024*, CCS '24, page 2385–2399, New York, NY, USA, 2024. Association for Computing Machinery. `doi:10.1145/3658644.3690220`.

[Hal24] Dan Hall. Zero Trust WARP: tunneling with a MASQUE, March 2024. `https://blog.cloudflare.com/zero-trust-warp-with-a-masque`.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HR24] Julia Hesse and Michael Rosenberg. PAKE combiners and efficient post-quantum instantiations. Cryptology ePrint Archive, Paper 2024/1621, 2024. URL: `https://eprint.iacr.org/2024/1621`.

[HSC$^+$23] Senyang Huang, Rui Qi Sim, Chitchanok Chuengsatiansup, Qian Guo, and Thomas Johansson. Cache-timing attack against HQC. *IACR TCHES*, 2023(3):136–163, 2023. `doi:10.46586/tches.v2023.i3.136-163`.

[IT21] J. Iyengar (Ed.) and M. Thomson (Ed.). QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000 (Proposed Standard), May 2021. URL: `https://www.rfc-editor.org/rfc/rfc9000.txt`, `doi:10.17487/RFC9000`.

[JMZ23] Haodong Jiang, Zhi Ma, and Zhenfeng Zhang. Post-quantum security of key encapsulation mechanism against CCA attacks with a single decapsulation query. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part IV*, volume 14441 of *LNCS*, pages 434–468. Springer, Singapore, December 2023. `doi:10.1007/978-981-99-8730-6_14`.

[JRX24] Jake Januzelli, Lawrence Roy, and Jiayu Xu. Under what conditions is encrypted key exchange actually secure? Cryptology ePrint Archive, Report 2024/324, 2024. URL: https://eprint.iacr.org/2024/324.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997. Updated by RFC 6151. URL: https://www.rfc-editor.org/rfc/rfc2104.txt, doi:10.17487/RFC2104.

[KE10] H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (Informational), May 2010. URL: https://www.rfc-editor.org/rfc/rfc5869.txt, doi:10.17487/RFC5869.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, Berlin, Heidelberg, August 2010. doi:10.1007/978-3-642-14623-7_34.

[LHT16] A. Langley, M. Hamburg, and S. Turner. Elliptic Curves for Security. RFC 7748 (Informational), January 2016. URL: https://www.rfc-editor.org/rfc/rfc7748.txt, doi:10.17487/RFC7748.

[LL24] You Lyu and Shengli Liu. Hybrid password authentication key exchange in the UC framework. Cryptology ePrint Archive, Paper 2024/1630, 2024. URL: https://eprint.iacr.org/2024/1630.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, Berlin, Heidelberg, February 2011. doi:10.1007/978-3-642-19074-2_21.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, Berlin, Heidelberg, May 2013. doi:10.1007/978-3-642-38348-9_3.

[Mat21] Nick Mathewson. Ntor protocol with extra data, version 3. https://spec.torproject.org/proposals/332-ntor-v3-with-extra-data.html, 2021.

[Moh10] Payman Mohassel. A closer look at anonymity and robustness in encryption schemes. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 501–518. Springer, Berlin, Heidelberg, December 2010. doi:10.1007/978-3-642-17373-8_29.

[MRR20] Ian McQuoid, Mike Rosulek, and Lawrence Roy. Minimal symmetric PAKE and 1-out-of-N OT from programmable-once public functions. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 425–442. ACM Press, November 2020. doi:10.1145/3372297.3417870.

[MX23] Varun Maram and Keita Xagawa. Post-quantum anonymity of Kyber. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 3–35. Springer, Cham, May 2023. doi:10.1007/978-3-031-31368-4_1.

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

[NIS23] NIST. Digital signature standard (dss), February 2023. FIPS 186-5. `https://doi.org/10.6028/NIST.FIPS.186-5`.

[PG25] Jonas Janneck Phillip Gajland, Vincent Hwang. Shadowfax: Combiners for deniability. Cryptology ePrint Archive, Paper 2025/154, 2025. URL: `https://eprint.iacr.org/2025/154`.

[Rad] Cloudflare Radar. Adoption & Usage Worldwide | Cloudflare Radar. URL: `https://radar.cloudflare.com/adoption-and-usage#post-quantum-encryption-adoption`.

[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. `doi:10.1145/1060590.1060603`.

[ROSW24] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. TLS Encrypted Client Hello. Internet Draft draft-ietf-tls-esni-22, Internet Engineering Task Force, September 2024. `https://datatracker.ietf.org/doc/draft-ietf-tls-esni-22`.

[SGJ23] Bruno Freitas Dos Santos, Yanqi Gu, and Stanislaw Jarecki. Randomized half-ideal cipher on groups with applications to UC (a)PAKE. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 128–156. Springer, Cham, April 2023. `doi:10.1007/978-3-031-30589-4_5`.

[sha23] Shadowsocks. `https://shadowsocks.org/doc/what-is-shadowsocks.html`, 2023.

[SP22] Benjamin M. Schwartz and Christopher Patton. The Pseudorandom Extension for cTLS. Internet Draft draft-cpbs-pseudorandom-ctls-01, Internet Engineering Task Force, April 2022. `https://datatracker.ietf.org/doc/draft-cpbs-pseudorandom-ctls-01`.

[SSL20] Sven Schäge, Jörg Schwenk, and Sebastian Lauer. Privacy-preserving authenticated key exchange and the case of IKEv2. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 567–596. Springer, Cham, May 2020. `doi:10.1007/978-3-030-45388-6_20`.

[The19] The Tor Project. obfs4 (the obfourscator), protocol specification, version c0898c2d. `https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transports/lyrebird/-/blob/main/doc/obfs4-spec.txt`, 2019.

[Tib14] Mehdi Tibouchi. Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 139–156. Springer, Berlin, Heidelberg, March 2014. `doi:10.1007/978-3-662-45472-5_10`.

[Vai21] Loup Vaillant. Elligator - Hiding Key Exchanges, 2021. URL: `https://elligator.org/key-exchange`.

[vme19] VMess. `https://www.v2ray.com/en/configuration/protocols/vmess.html`, 2019.

[Wes24] Bas Westerbaan. The state of the post-quantum Internet, March 2024. `https://blog.cloudflare.com/pq-2024`.

[WJJS23] Ryan Wails, Rob Jansen, Aaron Johnson, and Micah Sherr. Proteus: Programmable protocols for censorship circumvention. In *Free and Open Communication on the Internet*, number 2, pages 50–66, 2023.

[WMSM11] Andrew M. White, Austin R. Matthews, Kevin Z. Snow, and Fabian Monrose. Phonotactic reconstruction of encrypted VoIP conversations: Hookt on fon-iks. In *2011 IEEE Symposium on Security and Privacy*, pages 3–18. IEEE Computer Society Press, May 2011. `doi:10.1109/SP.2011.34`.

[WSS$^+$23] Mingshi Wu, Jackson Sippe, Danesh Sivakumar, Jack Burg, Peter Anderson, Xiaokang Wang, Kevin Bock, Amir Houmansadr, Dave Levin, and Eric Wustrow. How the great firewall of china detects and blocks fully encrypted traffic. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 2653–2670. USENIX Association, August 2023.

[WW24] Bas Westerbaan and Christopher A. Wood. X25519Kyber768Draft00 hybrid post-quantum KEM for HPKE. Internet Draft draft-westerbaan-cfrg-hpke-xyber768d00-03, Internet Engineering Task Force, May 2024. Num Pages: 20. URL: `https://datatracker.ietf.org/doc/draft-westerbaan-cfrg-hpke-xyber768d00`.

[Xag22] Keita Xagawa. Anonymity of NIST PQC round 3 KEMs. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 551–581. Springer, Cham, May / June 2022. `doi:10.1007/978-3-031-07082-2_20`.

[XAR$^+$24] Diwen Xue, Anna Ablove, Reethika Ramesh, Grace Kwak Danciu, and Roya Ensafi. Bridging barriers: A survey of challenges and priorities in the censorship circumvention landscape. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

A Additional Definitions

Definition A.1 (Pseudorandom generator). *We define the* pseudorandom generator *(*PRG*) advantage of an adversary $\mathcal{A}$ against a function $\mathsf{G}\colon \mathcal{S} \to \mathcal{R}$ as*

$$\mathsf{Adv}^{\mathsf{PRG}}_{\mathsf{G}}(\mathcal{A}) := \Pr\left[\mathcal{A}(r_1) \Rightarrow 1 \mid s \leftarrow_{\$} \mathcal{S}, r \leftarrow \mathsf{G}(s)\right] - \Pr\left[\mathcal{A}(r_0) \Rightarrow 1 \mid r \leftarrow_{\$} \mathcal{R}\right].$$

Definition A.2 (Pseudorandom function)**.** *We define the* pseudorandom function *(*PRF*) advantage of an adversary $\mathcal{A}$ against a function* $\mathsf{F}\colon \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ *as*

$$\begin{aligned}\mathsf{Adv}_{\mathsf{F}}^{\mathsf{PRF}}(\mathcal{A}) := \ & \Pr\left[\mathcal{A}^{\mathsf{F}(k,\cdot)}() \Rightarrow 1 \mid k \leftarrow_{\$} \{0,1\}^{\kappa}\right] \\ & - \Pr\left[\mathcal{A}^{R(\cdot)}() \Rightarrow 1 \mid R \leftarrow_{\$} \{\textit{all functions}\colon \mathcal{Y} \to \mathcal{Z}\}\right].\end{aligned}$$

Definition A.3 (One-time indistinguishability from random strings (OT-IND\$))**.** *Let* SE *be a symmetric encryption scheme with key space $\mathcal{K}$ and where the length $|c| = \ell(|m|)$ of the ciphertext output by encryption only depends on the input message's length $|m|$. We define the* OT-IND\$ *security game as follows.*

1. *The challenger samples $b \leftarrow_{\$} \{0,1\}$.*
2. *$\mathcal{A}$ provides a chosen message m to the challenger, who returns either* $\mathsf{SE.Enc}(m)$ *if $b = 1$ or a random string $c \leftarrow_{\$} \{0,1\}^{\ell(|m|)}$ if $b = 0$.*
3. *$\mathcal{A}$ outputs a guess $b' \in \{0,1\}$.*

We define the OT-IND\$ *advantage of an adversary $\mathcal{A}$ as*

$$\mathsf{Adv}_{\mathsf{SE}}^{\mathsf{OT\text{-}IND\$}}(\mathcal{A}) := 2 \cdot \Pr[b' = b] - 1.$$

Definition A.4 (nm-PRF-ODH assumption)**.** *Let $\mathbb{G}$ be a group of prime order q with generator g, and* $\mathsf{F}\colon \mathbb{G} \times \{0,1\}^* \to \{0,1\}^{\ell}$ *for $\ell \in \mathbb{N}$ be a function.*

We define the nm-PRF-ODH *security game as follows.*

1. *The challenger samples $b \leftarrow_{\$} \{0,1\}$, $u, v \leftarrow_{\$} \mathbb{Z}_q$, and provides $\mathbb{G}$, g, g^u, and g^v to $\mathcal{A}$, who responds with a challenge label x^*. $\mathcal{A}$ has access to an oracle $\mathrm{ODH}_v(\cdot,\cdot)$ that on input $S \in \mathbb{G}, x \in \{0,1\}^*$ returns $y \leftarrow \mathsf{F}(S^v, x)$.*
2. *The challenger computes $y_0 = \mathsf{F}(g^{uv}, x^*)$ and samples $y_1 \leftarrow_{\$} \{0,1\}^{\ell}$ uniformly at random, providing y_b to $\mathcal{A}$.*
3. *$\mathcal{A}$ again has access to oracle $\mathrm{ODH}_v(\cdot,\cdot)$ that on input $S \in \mathbb{G}, x \in \{0,1\}^*$ returns $y \leftarrow \mathsf{F}(S^v, x)$, except that it disallows the input $(S, x) = (g^u, x^*)$.*
4. *Eventually, $\mathcal{A}$ stops and outputs a guess $b' \in \{0,1\}$.*

We define the nm-PRF-ODH *advantage function as*

$$\mathsf{Adv}_{\mathsf{F},\mathbb{G}}^{\mathsf{nm\text{-}PRF\text{-}ODH}}(\mathcal{A}) := 2 \cdot \Pr[b' = b] - 1.$$

B Relations Between Anonymity Notions

Several of the security properties that we have proven for our obfuscated KEM combiner — strong pseudorandomness (SPR-CCA), ciphertext uniformity, and public key uniformity — relate to existing notions of anonymity and pseudorandomness of KEMs. We compare the goals of these notions and discuss how they relate to one another. [JRX24, Table 3] provides a nice summary of PAKE-related notions of pseudorandomness and anonymity. We reiterate their observations and extend them to include additional properties.

```
$\mathsf{G}^{\mathsf{ANO\text{-}CCA}}_{\mathsf{K}}(\mathcal{A})$:
1 $b \leftarrow\!\!\$ \{0,1\}$
2 $(pk_0, sk_0) \leftarrow\!\!\$ \mathsf{KGen}()$
3 $(pk_1, sk_1) \leftarrow\!\!\$ \mathsf{KGen}()$
4 $(c^*, K^*) \leftarrow\!\!\$ \mathsf{Encap}(pk_b)$
5 $b' \leftarrow \mathcal{A}^{\mathsf{Decap}_{c^*}(sk_b,\cdot)}(pk_0, pk_1, (c^*, K^*))$
6 return $[\![b = b']\!]$

$\mathsf{Decap}_x(sk, c)$:
7 if $c = x$ then return $\perp$
8 $K \leftarrow \mathsf{Decap}(sk, c)$
9 return $K$
```

Figure 8: Security game for ANO-CCA of a KEM $\mathsf{K} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap})$.

The *strong pseudorandomness* (SPR-CCA) property that we have shown for our KEM combiner originates in [Xag22] as a tool for proving anonymity of a KEM. In particular, [Xag22, Theorem 2.5] shows that SPR-CCA implies ANO-CCA (which we recap in Figure 8), where ANO-CCA of a KEM captures the idea that an adversary cannot distinguish which public key corresponds to a given ciphertext, shared secret pair. This definition of anonymity for KEMs was introduced in [GMP22] and an analogous definition is given for PKE schemes. The ANO-CCA property for PKE schemes from [GMP22] is equivalent to the IK-CCA property of PKE schemes introduced in [BBDP01]. IK-CCA, also called "key-privacy" or "indistinguishability of keys," similarly captures the idea that an adversary cannot distinguish which of two public keys corresponds to a given challenge ciphertext. This idea of anonymity of PKE schemes is, again, similarly introduced by Mohassel [Moh10] for *general encryption schemes* (capturing PKE and IBE schemes). The definition in [Moh10] is similar to that of ANO/IK-CCA, except it also provides the adversary access to a number of public keys, and some secret keys that do not allow for trivial wins. This is equivalent to the ANO/IK-CCA definition up to a factor depending on the number of secret key reveals [Moh10]. Finally, all of these definitions of anonymity for PKE schemes are related to the *ciphertext anonymity* property introduced in [DHRR22a, Definition 14] for updatable randomizable PKE schemes (note that the text in the introduction and Section 5 refers to this property as *ciphertext anonymity*, while in the definition it is called *anonymity*). This definition of anonymity is a natural analogue of ANO/IK-CCA in the setting of updatable randomizable PKE schemes (i.e., additionally allows the adversary access to public keys and an oracle which re-randomizes public keys).

It follows directly from these results that our OKEM combiner provides ANO-CCA security (Figure 8). Therefore, instantiating our OKEM combiner with a DH-based OKEM and a post-quantum OKEM provides the first *hybrid anonymous KEM*. That is, the anonymity of the scheme relies on the stronger underlying security assumption of the input KEMs. Prior hybrid KEMs, e.g., X-Wing [BCD+24] and those resulting from the generic constructions in [GHP18], do not achieve such a hybrid anonymity property because they simply concatenate the two ciphertexts before returning them in Encap. This means that being able to correlate one of the ciphertexts with its corresponding public key is enough to break the anonymity of the scheme. For example, since the anonymity of ML-KEM relies on an underlying module LWE assumption [MX23], then breaking this assumption would be sufficient to violate the anonymity of any prior hybrid construction instantiated with ML-KEM.

The notions of *public key uniformity* and *ciphertext uniformity* that we adopt from [GSV24] are similar to existing definitions from the literature on password-authenticated key exchange (PAKE). In particular, [SGJ23] says that a KEM has *uniform public keys* if the distribution of public keys output by KGen is indistinguishable from uniformly sampling from the public key space. This is equal to the *fuzziness* property of KEMs given in [BCP+23].

Definition B.1 (KEM Fuzziness [BCP+23])**.** *Let* $\mathsf{K} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap})$ *be a KEM with public key space* $\mathcal{P}$*. The advantage of an unbounded adversary* $\mathcal{A}$ *in breaking the fuzziness of* K *is:*

$$\mathsf{Adv}_{\mathsf{K}}^{\mathsf{fuzzy}}(\mathcal{A}) := 2 \cdot \Pr\left[\mathcal{A}(pk_b) = b \;\middle|\; \begin{array}{c} b \leftarrow_\$ \{0,1\}, pk_0 \leftarrow_\$ \mathcal{P}, \\ (sk_1, pk_1) \leftarrow_\$ \mathsf{KGen}() \end{array}\right] - 1$$

Public key uniformity as defined in Definition 2.6 is equivalent to fuzziness when the public key space is the set of bitstrings. Similarly, [SGJ23] and [BCP+23] define properties of KEMs that are similar to ciphertext uniformity as defined in Definition 2.7. [BCP+23] says that a KEM is *anonymous* if the ciphertexts output by Encap are indistinguishable from uniformly sampled ciphertexts from the ciphertext space.

Definition B.2 (KEM Anonymity [BCP+23])**.** *Let* $\mathsf{K} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap})$ *be a KEM with ciphertext space* $\mathcal{C}$*. The advantage of an unbounded adversary* $\mathcal{A}$ *in breaking the anonymity of* K *is:*

$$\mathsf{Adv}_{\mathsf{K}}^{\mathsf{ano}}(\mathcal{A}) := 2 \cdot \Pr\left[\mathcal{A}(c_b) = b \;\middle|\; \begin{array}{c} b \leftarrow_\$ \{0,1\}, c_0 \leftarrow_\$ \mathcal{C}, \\ (sk, pk) \leftarrow_\$ \mathsf{KGen}(), \\ (c_1, K_1) \leftarrow_\$ \mathsf{Encap}(pk) \end{array}\right] - 1$$

[SGJ23] says that a KEM is *anonymous* if two KEM ciphertexts output by Encap on two randomly generated public keys are indistinguishable. This is a slightly weaker property than the anonymity definition from [BCP+23]. Ciphertext uniformity requires that ciphertexts output by Encap are indistinguishable from uniform bitstrings and also provides the adversary with the KEM public key (and in the case of strong ciphertext uniformity, the secret key), so ciphertext uniformity is a stronger notion than KEM anonymity in both [BCP+23, SGJ23]. The relations between these notions are summarized in Figure 3

Other similar notions of anonymity for key agreement and key exchange protocols include but are not limited to *key-hiding AKEs* [GJK21], *privacy-preserving AKEs* [SSL20], *pseudorandom (unauthenticated) key agreement* protocols [MRR20], the *random-message* property of [SGJ23], *one-way anonymity* [GSU13], and *anonymous ratcheted key exchange* [DHRR22b]. As these definitions apply to protocols rather than primitives, we do not include them in our comparison.

C Security Proofs for OEINC

C.1 IND-CPA / IND-CCA Security

We first provide the proof of Theorem 3.1 (IND-CPA / IND-CCA security of OEINC). We focus on the IND-CCA case in our proof. The IND-CPA case follows analogously, merely omitting the handling of decapsulation queries.

Proof. **Game 0.** We start with the security game for IND-CCA, $\mathsf{G}_{\mathsf{OEINC}}^{\mathsf{IND\text{-}CCA}}(\mathcal{A})$.

Case I.

We begin with the first case reducing to IND-CCA security of the outer OKEM, and proceed via a series of game hops.

Game I.0. In $\mathsf{G}_{\mathrm{I.0}}$, we replace the key K_{out} with a uniformly random $\widetilde{K}_{out}$.

We bound the adversary $\mathcal{A}$'s advantage by a reduction $\mathcal{B}_1$ to the $\mathsf{IND\text{-}CCA}$ security of $\mathsf{outOKEM}$. $\mathcal{B}_1$ obtains the $\mathsf{IND\text{-}CCA}$ challenge (pk, c^*, K^*) and simulates the game for $\mathcal{A}$ as follows. It uses pk, c^*, and K^* in place of the outer pk_{out}, c_{out}, and K_{out}, respectively. Upon any $\mathsf{Decap}(c = c_{out} \| c'_{in})$ query, $\mathcal{B}_1$ checks if $c_{out} = c^*$ and if so, replaces K_{out} with $\widetilde{K}_{out}$. Else, $\mathcal{B}_1$ queries its $\mathsf{IND\text{-}CCA}$ decapsulation oracle and uses the response as K_{out}.

If K^* is the real KEM key then $\mathcal{B}_1$ exactly simulates G_0 for $\mathcal{A}$; else, if K^* is random then $\mathcal{B}_1$ exactly simulates $\mathsf{G}_{\mathrm{I.0}}$. Therefore:

$$\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_{\mathrm{I.0}}] \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{outOKEM}}(\mathcal{B}_1).$$

Game I.1. In $\mathsf{G}_{\mathrm{I.1}}$ we replace the output K_{oe}, K_{ok} of $\mathsf{G}(\widetilde{K}_{out})$ with uniformly random $\widetilde{K}_{oe}, \widetilde{K}_{ok}$, in particular for the challenge. We bound the difference in this step by a reduction $\mathcal{B}_2$ to the PRG security of G. The reduction uses its oracle in place of G, simulating either $\mathsf{G}_{\mathrm{I.0}}$ or $\mathsf{G}_{\mathrm{I.1}}$, giving:

$$\Pr[\mathsf{G}_{\mathrm{I.0}}] - \Pr[\mathsf{G}_{\mathrm{I.1}}] \leq \mathsf{Adv}^{\mathsf{PRG}}_{\mathsf{G}}(\mathcal{B}_2).$$

Game I.2. Finally, in $\mathsf{G}_{\mathrm{I.2}}$, we replace evaluations of $\mathsf{W}(\widetilde{K}_{ok}, \cdot, \cdot)$ with a random function. This in particular replaces the challenge K with a uniformly random $\widetilde{K}$, independent of all keys returned by the decapsulation oracle because the third input c to W is distinct from the challenge c^* for each query. We bound this step by a reduction to the skPRF security of W, where the reduction $\mathcal{B}_3$ uses its oracle in place of $\mathsf{W}(\widetilde{K}_{ok}, \cdot, \cdot)$. This yields:

$$\Pr[\mathsf{G}_{\mathrm{I.1}}] - \Pr[\mathsf{G}_{\mathrm{I.2}}] \leq \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{W},1}(\mathcal{B}_3).$$

We now have in $\mathsf{G}_{\mathrm{I.2}}$ that the real and random KEM key output of the $\mathsf{IND\text{-}CCA}$ game are both randomly sampled, and so $\mathcal{A}$ has no better chance than guessing the challenge bit b:

$$\mathsf{Adv}^{\mathsf{G}_{\mathrm{I.2}}}_{\mathsf{OEINC}}(\mathcal{A}) = 0.$$

Case II.

This case reduces to the $\mathsf{IND\text{-}CCA}$ security of the inner OKEM, again proceeding via a series of game hops.

Game II.0. In $\mathsf{G}_{\mathrm{II.0}}$, we replace the key K_{in} with a uniformly random $\widetilde{K}_{in}$.

We bound the adversary $\mathcal{A}$'s difference in advantage by a reduction $\mathcal{C}_1$ to the $\mathsf{IND\text{-}CCA}$ security of inOKEM. $\mathcal{C}_1$ obtains the $\mathsf{IND\text{-}CCA}$ challenge (pk, c^*, K^*) and simulates the game for $\mathcal{A}$ as follows. It uses pk, c^*, and K^* in place of the inner pk_{in}, c_{in}, and K_{in}, respectively. Upon decapsulation queries from $\mathcal{A}$, $\mathcal{C}_1$ checks if (in line 20 of Decap) $c_{in} = c^*$ and if so, uses $\widetilde{K}_{in}$ as K_{in}. Else, it queries its $\mathsf{IND\text{-}CCA}$ decapsulation oracle and uses the response as K_{in}. If K^* is the real KEM key than $\mathcal{C}_1$ exactly simulates G_0 to $\mathcal{A}$; else, it simulates $\mathsf{G}_{\mathrm{II.0}}$. Thus:

$$\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_{\mathrm{II.0}}] \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{inOKEM}}(\mathcal{C}_1).$$

Game II.1. Lastly in $\mathsf{G}_{\text{II.1}}$, we replace evaluations of $\mathsf{W}(\cdot, K_{in}, \cdot)$ with a random function. This replaces K with a uniformly random $\widetilde{K}$. Note that on any decapsulation query, the third input c to W is distinct from the challenge c^*. We bound this by a reduction $\mathcal{C}_2$ to the skPRF security of W, where $\mathcal{C}_2$ uses its oracle in place of calls to $\mathsf{W}(\cdot, K_{in}, \cdot)$. Therefore:

$$\Pr[\mathsf{G}_{\text{II.0}}] - \Pr[\mathsf{G}_{\text{II.1}}] \leq \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{W},2}(\mathcal{C}_2).$$

Now, as before, the real and random KEM key output of the $\mathsf{IND\text{-}CCA}$ game are both randomly sampled, and so $\mathcal{A}$ has no better chance than guessing the challenge bit b:

$$\mathsf{Adv}^{\mathsf{G}_{\text{II.1}}}_{\mathsf{OEINC}}(\mathcal{A}) = 0.$$

Collecting the bounds yields the theorem statement. □

C.2 Public Key Uniformity

We give the proof of Theorem 3.4 ($\mathsf{pk\text{-}unif}$ of OEINC).

Proof. **Game 0.** We start with the security game for $\mathsf{pk\text{-}unif}$ ($\mathsf{G}^{\mathsf{pk\text{-}unif}}_{\mathsf{OEINC}}(\mathcal{A})$).

Game 1. In G_1 we replace the outer public key pk_{out} with a uniformly random string from $\{0,1\}^{\mathsf{ol}_{\mathsf{outOKEM}}}$. This game hop is bounded by a reduction $\mathcal{B}_1$ to $\mathsf{pk\text{-}unif}$ of $\mathsf{outOKEM}$. We have that

$$\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_1] \leq \mathsf{Adv}^{\mathsf{pk\text{-}unif}}_{\mathsf{outOKEM}}(\mathcal{B}_1).$$

Game 2. Now, in G_2, we replace pk_{in} with a uniformly random string from $\{0,1\}^{\mathsf{ol}_{\mathsf{inOKEM}}}$, bounding by a reduction $\mathcal{B}_2$ to the $\mathsf{pk\text{-}unif}$ security of inOKEM:

$$\Pr[\mathsf{G}_1] - \Pr[\mathsf{G}_2] \leq \mathsf{Adv}^{\mathsf{pk\text{-}unif}}_{\mathsf{inOKEM}}(\mathcal{B}_2).$$

Now, pk is a random pair from $\{0,1\}^{\mathsf{ol}_{\mathsf{outOKEM}}} \times \{0,1\}^{\mathsf{ol}_{\mathsf{inOKEM}}}$, so $\mathcal{A}$ cannot win anymore and we have

$$\mathsf{Adv}^{\mathsf{G}_2}_{\mathsf{OEINC}}(\mathcal{A}) = 0.$$ □

D An OKEM from DHKEM

We provide a construction of an obfuscated KEM based on DHKEM, as defined in the IETF HPKE standard [BBLW22], when instantiated with a prime-order curve, e.g., P-256 or Ristretto [NIS23, LHT16],[8] and the Elligator2 encoding. Then, we demonstrate that this achieves the required security properties to be of use in our OKEM combiner.

Consider the OKEM constructed via applying Elligator2 to DHKEM public keys and ciphertexts, following the keygen/encapsulate-then-encode paradigm defined in [GSV24]. We denote this construction DHKEM-Ell2. Specifically, we use the algorithms described in Figure 9. In DHKEM, the ciphertext output by Encap is in fact a public key of the underlying DH group. Therefore, the Elligator2 map applies to both public keys and ciphertexts in the construction.

[8]Really, any ECDH scheme can work, so long as its public keys are uniformly distributed. This is not the case for, e.g., X25519 [Ber06], since scalar *clamping* ensures that public keys are in the prime-order subgroup, which is only 1/8 of the curve. There exist modifications to X25519 key generation to recover uniformity of public keys [Vai21].

```
KGen():
repeat
    $sk \leftarrow\$ [1, p-1]$
    $pk \leftarrow g^{sk}$
    $\hat{pk} \leftarrow \mathsf{Elligator2.Encode}(pk)$
until $\hat{pk} \neq \bot$
return $(sk, \hat{pk})$
```

```
Decap($sk_R, \widehat{c}$):
$c \leftarrow \mathsf{Elligator2.Decode}(\widehat{c})$
$pk_E \leftarrow \mathsf{DeserializePublicKey}(c)$
$dh \leftarrow {pk_E}^{sk_R}$
$pk_{Rm} \leftarrow \mathsf{SerializePublicKey}(g^{sk_R})$
$kem\_context \leftarrow c \| pk_{Rm}$
$K \leftarrow \mathsf{G}(dh, kem\_context)$
return $K$
```

```
Encap($pk_R$):
repeat
    $sk_E \leftarrow\$ [1, p-1]$
    $pk_E \leftarrow g^{sk_E}$
    $dh \leftarrow {pk_R}^{sk_E}$
    $c \leftarrow \mathsf{SerializePublicKey}(pk_E)$
    $pk_{Rm} \leftarrow \mathsf{SerializePublicKey}(pk_{Rm})$
    $kem\_context \leftarrow c \| pk_{Rm}$
    $K \leftarrow \mathsf{G}(dh, kem\_context)$
    $\widehat{c} \leftarrow\$ \mathsf{Elligator2.Encode}(c)$
until $\widehat{c} \neq \bot$
return $(\widehat{c}, K)$
```

Figure 9: DHKEM-Ell2: Keygen/Encapsulate-then-encode OKEM from DHKEM and Elligator2, where DHKEM is defined over a nominal group $\mathbb{G}$ (cf. [ABH+21]) of prime-order p with generator g. G is a key derivation function.

We can then apply the following results for SPR-CCA and IND-CCA security of the resulting keygen/encapsulate-then-encode OKEM.

Theorem D.1 (Keygen/encapsulate-then-encode obfuscated KEM IND-CCA security [GSV24])**.** *Let* OKEM *be a keygen/encapsulate-then-encode obfuscated KEM based on a regular KEM* KEM*. For any adversary $\mathcal{A}$ against the* IND-CCA *security of* OKEM*, we give an algorithm $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{OKEM}}(\mathcal{A}) \leq 1/\epsilon^{\mathsf{1kgensucc}}_{\mathsf{OKEM}} \cdot 1/\epsilon^{\mathsf{1encsucc}}_{\mathsf{OKEM}} \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{B}).$$

Theorem D.2 (Keygen/encapsulate-then-encode obfuscated KEM SPR-CCA security [GSV24])**.** *Let* OKEM *be a keygen/encapsulate-then-encode obfuscated KEM based on a regular KEM* KEM*. For any adversary $\mathcal{A}$ against the* SPR-CCA *security of* OKEM*, we give algorithms $\mathcal{B}_1$, $\mathcal{B}_2$ such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{OKEM}}(\mathcal{A}) \leq 1/\epsilon^{\mathsf{1kgensucc}}_{\mathsf{OKEM}} \cdot 1/\epsilon^{\mathsf{1encsucc}}_{\mathsf{OKEM}} \cdot \mathsf{Adv}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{KEM},\mathcal{S}_{\mathsf{unif}}}(\mathcal{B}_1)\\ + \mathsf{Adv}^{\mathsf{reg\text{-}ctxt\text{-}unif}}_{\mathsf{OKEM}}(\mathcal{B}_2),\end{aligned}$$

where $\mathcal{S}_{\mathsf{unif}}$ samples a fresh key pair (sk, pk) and outputs the ciphertext resulting from encapsulating against pk.

The terms relating to success probability of encoding ($\epsilon^{\mathsf{1kgensucc}}_{\mathsf{DHKEM\text{-}Ell2}}$, $\epsilon^{\mathsf{1encsucc}}_{\mathsf{DHKEM\text{-}Ell2}}$), are both $\approx 2^{-1}$, following from the fact that approximately half of the x-coordinates of input curves are quadratic residues. The ciphertext uniformity, $\mathsf{Adv}^{\mathsf{reg\text{-}ctxt\text{-}unif}}_{\mathsf{DHKEM\text{-}Ell2}}(\mathcal{A})$, and public key uniformity, $\mathsf{Adv}^{\mathsf{pk\text{-}unif}}_{\mathsf{DHKEM\text{-}Ell2}}(\mathcal{A})$, similarly depend on the parameters of the underlying curve and can be computed as was done in [GSV24, §2.1] for X25519 (with aforementioned keygen modifications). Furthermore, IND-CCA was proven for DHKEM in the context of DHIES [ABR01].[9] Therefore, it only remains to show SPR-CCA security of DHKEM. We do so in

[9] More precisely, the DH-based KEM used in DHIES resembles DHKEM, but with less domain separation in its KDF invocations and no input or output validation for public keys and shared secrets.

the following, via the $\mathsf{PRF\text{-}ODH}$ assumption [BFGJ17] on the KDF and group employed in DHKEM. Specifically, we use the $\mathsf{nm\text{-}PRF\text{-}ODH}$ assumption which allows multiple ODH_v oracle queries (to the group element that will correspond to the DHKEM public key), which we recap in Definition A.4.[10]

Theorem D.3 ($\mathsf{SPR\text{-}CCA}$ Security of DHKEM)**.** *Let* $\mathsf{DHKEM} = \mathsf{DHKEM}[\mathsf{P}, \mathsf{G}]$ *be the KEM described in [BBLW22] over a group* $\mathbb{G}$ *and key derivation function* G*. For any adversary* $\mathcal{A}$ *against the* $\mathsf{SPR\text{-}CCA}$ *security of* DHKEM*, we give an algorithm* $\mathcal{B}_1$ *such that*

$$\mathsf{Adv}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{DHKEM},\mathcal{S}_{\mathsf{unif}}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{nm\text{-}PRF\text{-}ODH}}_{\mathsf{G},\mathbb{G}}(\mathcal{B}_1)$$

where $\mathcal{S}_{\mathsf{unif}}$ *samples and outputs a random group element.*

Proof. **Game 0.** We start with the $\mathsf{SPR\text{-}CCA}$ security game $\mathsf{G}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{DHKEM},\mathcal{S}_{\mathsf{unif}}}(\mathcal{A})$.

Game 1. In G_1, we replace the real challenge key K_1^* with a randomly sampled key $\widetilde{K}$ from $\mathcal{K}$. We bound the difference by a reduction $\mathcal{B}_1$ to the $\mathsf{nm\text{-}PRF\text{-}ODH}$ security of G over $\mathbb{G}$. Note that in DHKEM, the ciphertext is simply an ephemeral group element g^r. The reduction obtains a $\mathsf{nm\text{-}PRF\text{-}ODH}$ challenge of the form (g^u, g^v, y_b^*), where y_b^* is either a random value or $\mathsf{G}(g^{uv}, x^*)$ for a challenge label x^*. Therefore, the reduction queries as label x^* the KEM context (g^u, g^v) as per [BBLW22] and uses its $\mathsf{nm\text{-}PRF\text{-}ODH}$ challenge y_b^* in place of K^*, g^u in place of c^*, and g^v in place of pk. Upon $\mathsf{Decap}(c)$ queries from $\mathcal{A}$, the reduction queries its $\mathsf{nm\text{-}PRF\text{-}ODH}$ oracle ODH_v on group element c and label (c, g^v) to obtain the KEM shared secret. When the challenge is real, $\mathcal{B}_1$ exactly simulates G_0 to $\mathcal{A}$; else, $\mathcal{B}_1$ exactly simulates G_1 to $\mathcal{A}$. It follows that

$$\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_1] \leq \mathsf{Adv}^{\mathsf{nm\text{-}PRF\text{-}ODH}}_{\mathsf{G}}(\mathcal{B}_1).$$

Game 2. In G_2, we replace the challenge ciphertext with a ciphertext $c^* \leftarrow_\$ \mathcal{S}_{\mathsf{unif}}$, i.e., a randomly sampled group element. Since the key $\widetilde{K}$ is random by G_1, the challenge ciphertext is no longer used to compute the key in Encap. Therefore, this change is unnoticeable to the adversary.

$$\Pr[\mathsf{G}_1] = \Pr[\mathsf{G}_2].$$

At this point, we have that c and K are both random. Hence, the adversary $\mathcal{A}$ can only guess the challenge bit b:

$$\mathsf{Adv}^{\mathsf{G}_2}_{\mathsf{DHKEM},\mathcal{S}_{\mathsf{unif}}}(\mathcal{A}) = 0. \qquad \square$$

Alternative constructions. A classically secure OKEM can also be constructed using DHKEM over elliptic curves in combination with $\mathsf{Elligator}^2$ [Tib14] in a similar manner. Compared to $\mathsf{Elligator2}$, the $\mathsf{Elligator}^2$ construction has 100% success probability. Naturally, substituting different encodings and/or elliptic curve groups alters the bounds accordingly.

[10]The $\mathsf{nm\text{-}PRF\text{-}ODH}$ assumption essentially encodes modeling the KDF as a random oracle and assuming the strong or gap Diffie–Hellman problem is hard in the underlying group.

E Obfuscated Key Exchange Security Model

We repeat the key exchange model introduced in [GSV24]. We specify a key exchange protocol KE through two algorithms:

- $\underline{\mathsf{Setup}}(id, role) \xrightarrow{\$} (sk, pk, st)$ generates the public-secret key pair (pk, sk) as well as the initial user state st for a protocol user with identity id in role $role \in \{\mathsf{initiator}, \mathsf{responder}\}$.
- $\underline{\mathsf{Run}}(\pi_u^i, st_u, sk_u, pk_v, m) \xrightarrow{\$} (\pi_u^i, st_u, m')$ processes a protocol message m delivered to session π_u^i following the protocol specification (along with inputs the session owner's state st_u and secret key sk_u, and the session's peer public key pk_v), updates π_u^i and st_u accordingly, and outputs the response message m'.

Further, we write $\mathsf{KE.KS}$ to denote the session key space of KE.

E.1 Session and game variables.

Session object π_u^i captures the session information for the ith session owned by user u. Each user u is assigned a role, $u.role \in \{\mathsf{initiator}, \mathsf{responder}\}$, and acts accordingly as initiator or responder in the protocol.

Each session object π_u^i holds several variables. The following session variables in *italics* font are accessible by the key exchange protocol (i.e., Run):

- $\pi_u^i.peerid$: the identity of the session's intended peer.
- $\pi_u^i.status$: the state of execution (initially $\mathsf{running}$, then set once by the protocol to $\mathsf{accepted}$ or $\mathsf{rejected}$).
- $\pi_u^i.skey$: the session key.
- $\pi_u^i.sid$, $\pi_u^i.cid$: the session and contributive identifiers.

These following session variables in sans-serif font are accessible by the security game only:

- $\pi_u^i.\mathsf{t_{acc}}$: the time at which the session accepted (initially ∞).
- $\pi_u^i.\mathsf{revealed}$, $\pi_u^i.\mathsf{tested}$: flags indicating whether the session was revealed or tested, respectively.

The security game further tracks the following game variables:

- time: a logical clock to order queries by the adversary.
- users: the number of users in the game.
- b: the challenge bit.
- sk_u, pk_u: the secret and public key of user u.
- revsk_u, revpk_u: flags indicating whether the secret or public, key of user u was revealed.

For syntactical convenience, we will interpret variables which are unset or set to ∞ as false in boolean conditions.

E.2 Session identifiers, contributive identifiers, and partnering.

We use *session identifiers* [BPR00] to determine that two sessions π_u^i, π_v^j are *partnered* if and only if $\pi_u^i.sid = \pi_v^j.sid$. Partnering is used to define basic correctness/soundness properties and to exclude trivial attacks like testing and revealing two partnered sessions that jointly executed the protocol.

We further use *contributive identifiers* [DFGS15] to determine when a responder session has an honest communication partner. Recall that initiators are unauthenticated. Hence, to avoid trivial attacks, the adversary may test a responder session only if that session honestly received the values specified in the contributive identifier *cid*.

E.3 Security Definition

Definition E.1 (Obfuscated key exchange security)**.** *Let* KE *be a key exchange protocol and* $\mathsf{G}^{\mathsf{ObfKE}}_{\mathsf{KE},\mathcal{S}}(\mathcal{A})$ *be the obfuscated key exchange security game wrt. a simulator* $\mathcal{S}$ *defined in Figure 10 for an adversary* $\mathcal{A}$*. We define the advantage of* $\mathcal{A}$ *in breaking the* ObfKE *security of* KE *as*

$$\mathsf{Adv}^{\mathsf{ObfKE}}_{\mathsf{KE},\mathcal{S}}(\mathcal{A}) := 2 \cdot \Pr\left[\mathsf{G}^{\mathsf{ObfKE}}_{\mathsf{KE},\mathcal{S}}(\mathcal{A}) \Rightarrow 1\right] - 1.$$

We distinguish two flavors of ObfKE*, capturing regular (*rObfKE*) and strong (*sObfKE*) obfuscation through different* $\mathsf{ObfFresh}$ *predicates (Figure 11); we omit the prefixes if the flavor is clear from context.*

E.4 Single-challenge selective security

For our security analysis, we will establish security in a simpler version of the obfuscated key exchange game, where the adversary has to commit to winning via either:

1. a single TEST query to some pre-declared session, or
2. a single CHALLEXEC query against some pre-declared server.

This variant is denoted as $\mathsf{ObfKE\text{-}1}$, and is a weaker versions of the main game, asking only for selective security in a single-challenge setting. It has been shown in [GSV24] that this simpler version generically implies full security (per Definition E.1) via a hybrid argument, for the same type of obfuscation (regular or strong). The hybrid works via guessing which session or server the adversary will challenge, losing a factor $(n_s + n_r q_{\mathrm{C}})$ when reducing ObfKE to $\mathsf{ObfKE\text{-}1}$ security, where n_s and n_r are the number of sessions and servers in the game and q_{C} the number of CHALLEXEC queries the adversary makes.

F Security Proof for `Drivel` (Theorem 4.1)

We proceed via a series of game hops. The majority of the proof directly follows the proof of security of `pq-obfs` from [GSV24]; we recap these steps here, in part verbatim. Game hops with significant differences to `pq-obfs` are underlined. The primary difference is that, in the final steps, OT-IND\$ is applied in $\mathsf{G}_{\mathrm{II}.6}$ and $\mathsf{G}_{\mathrm{II}.7}$ rather than invoking public key and ciphertext uniformity.

$\mathsf{G}^{\mathsf{ObfKE}}_{\mathsf{KE},\mathcal{S}}(\mathcal{A})$

```
Initialize:
$\mathsf{time} \leftarrow 0$; $\mathsf{users} \leftarrow 0$
$\mathsf{b} \leftarrow\$ \{0,1\}$
$\mathsf{Probed} \leftarrow \mathsf{false}$
```

```
NewUser($id, role$):
$u \leftarrow {+}{+}\mathsf{users}$
$u.role \leftarrow role$
$(pk_u, sk_u, st_u) \leftarrow\$ \mathsf{Setup}(id, role)$
$\mathsf{revsk}_u \leftarrow \infty$ ; $\mathsf{revpk}_u \leftarrow \infty$
```

```
RevSessionKey($u, i$):
if $\pi^i_u = \bot$ or $\neg\pi^i_u.\mathsf{t_{acc}}$ then return $\bot$
$\pi^i_u.\mathsf{revealed} \leftarrow \mathsf{true}$
return $\pi^i_u.skey$
```

```
RevPublicKey($u$):
if $\mathsf{revpk}_u$ then return $\bot$
$\mathsf{revpk}_u \leftarrow {+}{+}\mathsf{time}$
return $pk_u$
```

```
RevSecretKey($u$):
if $\mathsf{revsk}_u$ then return $\bot$
$\mathsf{revsk}_u \leftarrow {+}{+}\mathsf{time}$
if $\neg\mathsf{revpk}_u$ then $\mathsf{revpk}_u \leftarrow \mathsf{time}$ // Consider $pk_u$ revealed, too
return $sk_u$
```

```
Test($u, i$):
if $\pi^i_u = \bot$ or $\neg\pi^i_u.\mathsf{t_{acc}}$ or $\pi^i_u.\mathsf{tested}$ then
    return $\bot$
$\pi^i_u.\mathsf{tested} \leftarrow \mathsf{true}$
$k_1 \leftarrow \pi^i_u.skey$
$k_0 \leftarrow\$ \mathsf{KE.KS}$
return $k_\mathsf{b}$
```

```
Finalize($\mathsf{b}'$):
if $\neg\mathsf{Sound}$ then $\mathsf{b}' \leftarrow \mathsf{b}$
if $\neg\mathsf{ExplicitAuth}$ then $\mathsf{b}' \leftarrow \mathsf{b}$
if $\mathsf{Probed}$ then $\mathsf{b}' \leftarrow \mathsf{b}$
if $\neg\mathsf{Fresh}$ then $\mathsf{b}' \leftarrow 0$
if $\neg\mathsf{ObfFresh}$ then $\mathsf{b}' \leftarrow 0$
return $[\![\mathsf{b} = \mathsf{b}']\!]$
```

```
Send($u, i, m$):
$f \leftarrow [\![\pi^i_u = \bot]\!]$ // First Send to this session?
if $\pi^i_u = \bot$ and $u.role = \mathsf{initiator}$ then
    $\pi^i_u.peerid \leftarrow m$; $m \leftarrow \varepsilon$ // "Virtual" first message to initiator sets peer id
if $\pi^i_u.status \in \{\mathsf{accepted}, \mathsf{rejected}\}$ then return $\bot$
$(\pi^i_u, st_u, m') \leftarrow\$ \mathsf{Run}(\pi^i_u, st_u, sk_u, pk_{\pi^i_u.peerid}, m)$
if $\pi^i_u.status = \mathsf{accepted}$ then $\pi^i_u.\mathsf{t_{acc}} \leftarrow {+}{+}\mathsf{time}$
if $f$ then // Upon first message sent/received...
    if $u.role = \mathsf{initiator}$ then
        $\mathcal{F}_{\pi^i_u.peerid} \leftarrow \mathcal{F}_{\pi^i_u.peerid} \cup \{m'\}$ // Record initators' first messages (per responder)
    if $u.role = \mathsf{responder}$ and $m' \neq \varepsilon$ then
        if $m \notin \mathcal{F}_u$ and $\neg\mathsf{revpk}_u$ then $\mathsf{Probed} \leftarrow \mathsf{true}$ // Response to a non–initiator-first message by a non–$pk$-revelead responder is a successful probe
        $\mathcal{F}_u \leftarrow \mathcal{F}_u \setminus \{m\}$ // Consider $m$ "consumed"
return $(\pi^i_u.status, m')$
```

```
ChallExec($u, v$):
$\pi_u \leftarrow \pi_v \leftarrow \bot$ // Temporary initator and responder sessions
if $u.role \neq \mathsf{initiator}$ or $v.role \neq \mathsf{responder}$ then return $\bot$
$trans_1 \leftarrow ()$; $\pi_u.peerid \leftarrow v$; $p \leftarrow u$; $m \leftarrow \varepsilon$
repeat // Execute entire protocol and collect real transcript
    $(\pi_p, st_p, m) \leftarrow\$ \mathsf{Run}(\pi_p, st_p, sk_p, pk_{\pi_p.peerid}, m)$
    $trans_1 \leftarrow trans_1 \| (m)$
    if $p = u$ then $p \leftarrow v$ else $p \leftarrow u$ // Switch parties
until $\pi_u.status = \pi_v.status = \mathsf{accepted}$
$trans_0 \leftarrow\$ \mathcal{S}()$ // Simulated transcript
$k_1 \leftarrow \pi_u.skey$; $k_0 \leftarrow\$ \mathsf{KE.KS}$ // Real-or-random session key
$\mathsf{chall}_v \leftarrow \mathsf{true}$ // Mark server $v$ as challenged
return $(trans_\mathsf{b}, k_\mathsf{b})$
```

Figure 10: Obfuscated key exchange security (ObfKE) game capturing key indistinguishability, explicit server authentication, obfuscation with respect to a simulator $\mathcal{S}$, and probing resistance, via predicates (Figure 11) Fresh, $\mathsf{ExplicitAuth}$, $\mathsf{ObfFresh}$, and flag Probed, respectively.

Fresh:

```
for each $\pi_u^i : \pi_u^i.\mathsf{tested}$
    if $\pi_u^i.\mathsf{revealed}$ then
        return $\mathsf{false}$ // tested session may not be revealed
    if $\exists \pi_v^j \neq \pi_u^i : \pi_v^j.sid = \pi_u^i.sid \wedge (\pi_v^j.\mathsf{tested} \vee \pi_v^j.\mathsf{revealed})$ then
        return $\mathsf{false}$ // tested session's partnered session may not be tested or revealed
    if $u.role = \mathsf{initiator} \wedge \mathsf{revsk}_{\pi_u^i.peerid} \leq \pi_u^i.\mathsf{t}_{\mathsf{acc}} \wedge \neg\exists \pi_v^j \neq \pi_u^i : (\pi_u^i.sid = \pi_v^j.sid)$
        return $\mathsf{false}$ // initiators: forward secrecy (peer's $sk$ unrevealed prior to acceptance) or passive execution
    if $u.role = \mathsf{responder} \wedge \neg\exists \pi_v^j \neq \pi_u^i : (\pi_u^i.cid = \pi_v^j.cid \wedge v.role = \mathsf{initiator})$
        return $\mathsf{false}$ // responders: security only for passive executions (as initiators are unauthenticated)
return $\mathsf{true}$
```

ObfFresh:

```
$\boxed{\text{if } \exists v : \mathsf{revpk}_v \wedge \mathsf{chall}_v \text{ then}}$ // Regular obfuscation (rObfKE)
$\dbox{\text{if } \exists v : \mathsf{revsk}_v \wedge \mathsf{chall}_v \text{ then}}$ // Strong obfuscation (sObfKE)
    return $\mathsf{false}$ // Challenge pk revealed (fwd-secret: prior to challenge)
return $\mathsf{true}$
```

ExplicitAuth:

```
// Explicit authentication of responders to initiators
$\forall \pi_u^i$: $\big( u.role = \mathsf{initiator} \wedge \pi_u^i.\mathsf{t}_{\mathsf{acc}} < \mathsf{revsk}_{\pi_u^i.peerid}$
        $\implies \exists \pi_v^j : v = \pi_u^i.peerid \wedge \pi_u^i.sid = \pi_v^j.sid \big)$
```

Sound:

```
if $\exists$ distinct $\pi_u^i, \pi_v^j, \pi_w^k : \pi_u^i.sid = \pi_v^j.sid = \pi_w^k.sid \neq \perp$ then
    return $\mathsf{false}$ // no triple sid match
if $\exists \pi_u^i, \pi_v^j : \pi_u^i.sid = \pi_v^j.sid \neq \perp \wedge u.role = v.role$ then
    return $\mathsf{false}$ // partnering implies different roles
if $\exists \pi_u^i, \pi_v^j : \pi_u^i.sid = \pi_v^j.sid \neq \perp \wedge \pi_u^i.cid \neq \pi_v^j.cid$ then
    return $\mathsf{false}$ // partnering implies same contributive identifiers
if $\exists \pi_u^i, \pi_v^j : \pi_u^i.sid = \pi_v^j.sid \neq \perp \wedge u.role = \mathsf{initiator} \wedge \pi_u^i.peerid \neq v$ then
    return $\mathsf{false}$ // partnering implies agreement on responder ID
if $\exists \pi_u^i, \pi_v^j : \pi_u^i.sid = \pi_v^j.sid \neq \perp \wedge \pi_u^i.skey \neq \pi_v^j.skey$ then
    return $\mathsf{false}$ // partnering implies same key
return $\mathsf{true}$
```

Figure 11: Predicates Fresh, ExplicitAuth, ObfFresh, and flag Probed for the obfuscated key exchange security (ObfKE) game (Figure 10) with regular ($\boxed{\text{regular (rObfKE)}}$) or strong ($\dbox{\text{strong (sObfKE)}}$) obfuscation.

Game 0. We start with the security game for strong obfuscation (sObfKE), $\mathsf{G}_0 = \mathsf{G}^{\mathsf{sObfKE}}_{\mathtt{Drivel},\mathcal{S}_{\mathtt{Drivel}}}$, omitting the $\mathcal{S}_{\mathtt{Drivel}}$ subscript from here on:

$$\mathsf{Adv}^{\mathsf{sObfKE}}_{\mathtt{Drivel}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{G}_0}_{\mathtt{Drivel}}(\mathcal{A}) = 2 \cdot \Pr[\mathsf{G}_0] - 1.$$

Game 1 (Exclude KEM public key collisions). We modify Game G_0 to abort if two honest sessions sample the same ephemeral KEM key (pk_e) or if two servers sample the same long-term KEM key (pk_S). This is bounded by the public key collision probability ($\mathsf{pkcoll}_{\mathsf{KEM}}, \mathsf{pkcoll}_{\mathsf{OKEM}}$) of the ephemeral and static KEM for the at most n_s ephemeral and n_r long-term KEM keys (Definition 2.4):

$$\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_1] \leq \mathsf{pkcoll}_{\mathsf{KEM}}(n_s) + \mathsf{pkcoll}_{\mathsf{OKEM}}(n_r).$$

Game 2 (KEM correctness). We modify Game G_1 to abort if for any keys $(sk, pk) \leftarrow_\$ \mathsf{KGen}()$ and encapsulation $(c, K) \leftarrow_\$ \mathsf{Encap}(pk)$, we have that $K \neq \mathsf{Decap}(sk, c)$. The probability of this happening for any tuple (sk, pk, c) is upper-bounded by the correctness error δ_{KEM}, δ_{OKEM} of the ephemeral and obfuscated KEM as defined in Definition 2.3. Since there are two such tuples per session, we can bound the probability of such an abort by

$$\Pr[\mathsf{G}_1] - \Pr[\mathsf{G}_2] \leq n_s \cdot (\delta_{\mathsf{KEM}} + \delta_{\mathsf{OKEM}}).$$

Establishing soundness (Sound). We observe that in Game G_2, the adversary cannot violate soundness anymore (i.e., the predicate Sound cannot be false). Checking the conditions in Sound (Figure 11), we have:

1. No triple sid match: For three session identifiers to match, two initiator sessions or two responder sessions would have to sample the same KEM keys. However, G_1 aborts in that case.

2. Partnering implies different roles: For two same-role sessions to be partnered, they would have to sample the same ephemeral KEM key. Again, G_1 aborts in this case.

3. Partnering implies same contributive identifiers: The contributive identifier contains a subset of entries in the session identifier. Hence, agreement on the latter implies agreement on the former.

4. Partnering implies agreement on responder ID: The session identifier contains the responder's long-term KEM key pk_S. As these do not collide by G_1, they uniquely identify the responder.

5. Partnering implies same key: For two partnered sessions to derive a different key, the inputs to the computation of *skey* would have to differ. This may occur if the inputs to Encap (or Decap) are not consistent or if Encap (or Decap) has a correctness error. The latter case is prevented by G_2. For the former, the elements in the session identifier uniquely determine the inputs; note in particular that it uniquely identifies the responder (as per above) and hence the *NodeID* input. Hence agreement on session identifiers implies that the same key is generated.

Game 3 (Prevent Probed being set). In Game G_3, we stop setting $\mathsf{Probed} \leftarrow \mathsf{true}$ (i.e., $\mathcal{A}$ cannot win anymore in G_3 by breaking probing resistance). We have

$$\Pr[\mathsf{G}_2] - \Pr[\mathsf{G}_3] \leq \Pr[\mathsf{G}_2 \text{ sets } \mathsf{Probed}].$$

Recall that for Probed to be set, a responder session, whose public key (pk_S, NodeID) is not revealed, must reply to a non-initiator-first message m, with a non-empty message m'. We will bound this probability via the following, branching game hops $\mathsf{G}_{2.1}$–$\mathsf{G}_{2.3}$.

Game 2.1 (Guess violated server and session). In Game $\mathsf{G}_{2.1}$, we guess a "target server" v^*, the first server (in order of creation) on which Probed is set to true, as well as the (first) session π^* in which this happens. We let the game abort if that guess is incorrect, i.e., if Probed is not set to true when the server responds in session π^* or if Probed is set to true on a response from a server that was created earlier. This introduces an according loss in the number of servers times the number of sessions:

$$\Pr[\mathsf{G}_3 \text{ sets } \mathsf{Probed}] \leq n_s \cdot n_r \cdot \Pr[\mathsf{G}_{2.1} \text{ sets } \mathsf{Probed}].$$

Game 2.2 (Random ES). In Game $\mathsf{G}_{2.2}$, we replace ES in any session with the target server v^* with a uniformly random value $\widetilde{ES}$.

We bound the difference in this step by a reduction to the PRF security of F_2. The reduction $\mathcal{B}_1$ does not sample NodeID for v^* itself, but instead uses its oracle to compute evaluations of $\mathsf{F}_2(\mathit{NodeID}, \cdot)$ in sessions with v^* by calling its oracle on K_S. When the oracle output is real, reduction $\mathcal{B}_1$ exactly simulates $\mathsf{G}_{2.1}$, whereas when the output is random, it simulates $\mathsf{G}_{2.2}$. Therefore, because NodeID is unrevealed for server v^*,

$$\Pr[\mathsf{G}_{2.1} \text{ sets } \mathsf{Probed}] - \Pr[\mathsf{G}_{2.2} \text{ sets } \mathsf{Probed}] \leq \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_2}(\mathcal{B}_1).$$

Game 2.3 (Random MAC_C). In Game $\mathsf{G}_{2.3}$, we replace any evaluation of F_1 on input random values $\widetilde{ES}$ in sessions with the target server v^* with a random function (per such value). This in particular replaces MAC_C in any session with v^* uniformly random values MAC^*_C.

We bound the difference in this step by a reduction to the PRF security of F_1. The reduction $\mathcal{B}_2$ does not sample $\widetilde{ES}$ in the target session π^* itself, but instead uses its oracle to compute $\mathsf{F}_1(\widetilde{ES}, \cdot)$; in particular for computing MAC_C. Based on the oracle's output, $\mathcal{B}_2$ correctly simulates either $\mathsf{G}_{2.2}$ or $\mathsf{G}_{2.3}$. Hence,

$$\Pr[\mathsf{G}_{2.2} \text{ sets } \mathsf{Probed}] - \Pr[\mathsf{G}_{2.3} \text{ sets } \mathsf{Probed}] \leq \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_1}(\mathcal{B}_2).$$

Observe that Probed in $\mathsf{G}_{2.3}$ is set if a non–initiator-first message m yields a reply by an unrevealed responder. Such a message m can be either (1) a replay of an already consumed message output by an honest initiator for this responder, or (2) one that is different from all honest initiators' first messages sent to this responder.

In the first case, m is rejected due to the replay check against the list of seen MAC values $st.\mathcal{S}_{MAC}$ recorded in the server's state.

In the second case, $m = epk_e \| c_S \| P_C \| M_C \| MAC_C$ must be different from any initiator session's message sent to this responder. By $\mathsf{G}_{2.3}$, the target client MAC value MAC^*_C that the session π^* with v^* that first sets probed is a random fl_1-bit value unknown to $\mathcal{A}$. The adversary $\mathcal{A}$ can therefore only guess MAC_C^* with probability

$$\Pr[\mathsf{G}_{2.3} \text{ sets } \mathsf{Probed}] \leq \frac{1}{2^{\mathsf{fl}_1}}.$$

This completes the branching game hops bounding

$$\Pr[\mathsf{G}_2 \text{ sets } \mathsf{Probed}] \leq n_s n_r \cdot \left(\mathsf{Adv}_{\mathsf{F}_2}^{\mathsf{PRF}}(\mathcal{B}_1) + \mathsf{Adv}_{\mathsf{F}_1}^{\mathsf{PRF}}(\mathcal{B}_2) + \frac{1}{2^{\mathsf{fl}_1}}\right).$$

We now continue with the main proof from G_3, where Probed is never set.

Game 4 (Prevent $\mathsf{ExplicitAuth}$ being violated). In Game G_4, we abort the game if $\mathsf{ExplicitAuth}$ is violated (i.e., $\mathcal{A}$ cannot win anymore in G_4 by breaking explicit authentication). We have

$$\Pr[\mathsf{G}_3] - \Pr[\mathsf{G}_4] \leq \Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_3].$$

Recall that for $\mathsf{ExplicitAuth}$ to be violated, an initiator session must accept with a peer whose secret key is uncompromised at this point in the game, but for which there is no session of that peer holding the same session identifier. Formally,

$$\Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_3] = \Pr\big[\exists \pi_u^i : u.role = \mathsf{initiator} \wedge \pi_u^i.\mathsf{t_{acc}} < \mathsf{revsk}_{\pi_u^i.peerid} \wedge \\ \forall \pi_v^j \text{ s.t. } v = \pi_u^i.peerid : \pi_u^i.sid \neq \pi_v^j.sid\big].$$

We will bound this probability again through a series of branching game hops $\mathsf{G}_{3.1}$–$\mathsf{G}_{3.6}$.

Game 3.1 (Guess violated initiator session and peer). In Game $\mathsf{G}_{3.1}$, we guess a "target session" π^*, the first session (in order of creation) which makes $\mathsf{ExplicitAuth}$ evaluate to false, as well as that session's peer, $v^* = \pi^*.peerid$. We let the game abort if that guess is incorrect, i.e., if $\mathsf{ExplicitAuth}$ is not violated when the session π^* accepts or if v^* is not its peer. This introduces an according loss in the number of sessions times the number of servers:

$$\Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_3] \leq n_s \cdot n_r \cdot \Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_{3.1}].$$

<u>Game 3.2 (Long-term KEM $\mathsf{IND\text{-}CCA}$).</u> Let

$$(c_S, K_S) \leftarrow\!\!\$\; \mathsf{OKEM.Encap}(pk_S)$$

be the encapsulation computed at session π^* with the long-term public key of v^*. In Game $\mathsf{G}_{3.2}$, we replace the long-term KEM key K_S with a uniformly random $\widetilde{K}_S$ in π^*. All values derived from K_S in π^* use the randomized value $\widetilde{K}_S$.

We bound the adversary $\mathcal{A}$'s difference in advantage by a reduction $\mathcal{B}_3$ to the $\mathsf{IND\text{-}CCA}$ security of OKEM. $\mathcal{B}_3$ obtains the $\mathsf{IND\text{-}CCA}$ challenge (pk, c^*, K^*) and simulates the game for $\mathcal{A}$ as follows. It uses pk as the long-term public key of v^*. In π^*, $\mathcal{B}_3$ uses c^* as the ciphertext c_S. In any session of v^*, if the ciphertext c_S received is not c^*, then $\mathcal{B}_3$ queries its $\mathsf{IND\text{-}CCA}$ decapsulation oracle and uses the response as K_S; else, if $c_S = c^*$, then $\mathcal{B}_3$ uses K^* as K_S. Note that by the definition of $\mathsf{ExplicitAuth}$, sk_S is not revealed to the adversary prior to π^* violating $\mathsf{ExplicitAuth}$ and thus $\mathcal{B}_3$ does not need to answer a RevSecretKey call on v^*. If K^* is the real KEM key then $\mathcal{B}_3$ has exactly simulated $\mathsf{G}_{3.1}$ to $\mathcal{A}$; else, if K^* is random, then $\mathcal{B}_3$ has exactly simulated $\mathsf{G}_{3.2}$ to $\mathcal{A}$. Therefore:

$$\Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_{3.1}] - \Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_{3.2}] \le \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{OKEM}}(\mathcal{B}_3).$$

Game 3.3 (Random ES). In Game $\mathsf{G}_{3.3}$, we replace ES with a uniformly random $\widetilde{ES}$ in π^*. We bound the difference in this step by a reduction to the $\mathsf{swap\text{-}PRF}$ security of F_2. The reduction $\mathcal{B}_4$ instead of sampling $\widetilde{K}_S$ uses its oracle to compute $\mathsf{F}_2(\cdot, \widetilde{K}_S)$ in π^*. When the output is real, reduction $\mathcal{B}_4$ exactly simulates $\mathsf{G}_{3.2}$, whereas when the output is random, it simulates $\mathsf{G}_{3.3}$. Therefore,

$$\Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_{3.2}] - \Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_{3.3}] \le \mathsf{Adv}^{\mathsf{swap\text{-}PRF}}_{\mathsf{F}_2}(\mathcal{B}_4).$$

Game 3.4 (Random ES'). In Game $\mathsf{G}_{3.4}$, we replace evaluations $\mathsf{F}_1(\widetilde{ES}, \cdot)$ in π^* with a random function. This in particular replaces ES' with a random value $\widetilde{ES'}$. We bound the difference in this step with the PRF security of F_1, via a reduction $\mathcal{B}_5$ using its oracle in place of $\mathsf{F}_1(\widetilde{ES}, \cdot)$:

$$\Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_{3.3}] - \Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_{3.4}] \le \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_1}(\mathcal{B}_5).$$

Game 3.5 (Random FS). In Game $\mathsf{G}_{3.5}$, we replace evaluations $\mathsf{F}_2(\widetilde{ES'}, \cdot)$ in π^* with a random function. This in particular replaces FS with a random value $\widetilde{FS}$. As in the previous hops, we can bound the difference by PRF security of F_2:

$$\Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_{3.4}] - \Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_{3.5}] \le \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_2}(\mathcal{B}_6).$$

Game 3.6 (Random $auth$). Finally, in Game $\mathsf{G}_{3.6}$, we replace evaluations $\mathsf{F}_1(\widetilde{FS}, \cdot)$ in π^* with a random function. This in particular replaces $auth$ with a random value $auth^*$, and again can be bounded by PRF security of F_1:

$$\Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_{3.5}] - \Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_{3.6}] \le \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_1}(\mathcal{B}_7).$$

Now, given that $auth^*$ is a uniformly random value unknown to $\mathcal{A}$, violating $\mathsf{ExplicitAuth}$ in π^* requires that $\mathcal{A}$ correctly guesses the fl_1-bit value of $auth^*$. The probability of $\mathcal{A}$ guessing correctly is

$$\Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_{3.6}] \le \frac{1}{2^{\mathsf{fl}_1}}.$$

This completes the branching, bounding

$$\begin{aligned}\Pr[\neg\mathsf{ExplicitAuth} \text{ in } \mathsf{G}_3] \le & n_s n_r \cdot \Big(\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{OKEM}}(\mathcal{B}_3) + \mathsf{Adv}^{\mathsf{swap\text{-}PRF}}_{\mathsf{F}_2}(\mathcal{B}_4) \\ & + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_1}(\mathcal{B}_5) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_2}(\mathcal{B}_6) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_1}(\mathcal{B}_7) + \frac{1}{2^{\mathsf{fl}_1}}\Big).\end{aligned}$$

We now continue with the main proof from Game G_4, where $\mathsf{ExplicitAuth}$ is never violated.

Game 5 (Single-challenge selective security). We now restrict the adversary to a single-challenge selective ($\mathsf{sObfKE\text{-}1}$) version G_5 of Game G_4, where $\mathcal{A}$ has to commit upfront to winning either

(I) via a single Test query (A = Test) on the pre-determined s-th created session, or

(II) via a single ChallExec query (A = ChallExec) on the pre-determined p-th created server.

Recall that in G_4, Sound is never violated, Probed is never set, and $\mathsf{ExplicitAuth}$ is never violated, hence $\mathcal{A}$ can only win via Test or ChallExec queries. Applying the hybrid argument from [GSV24, Theorem 4.3], we have

$$\mathsf{Adv}^{\mathsf{G}_4}_{\mathtt{Drivel}}(\mathcal{A}) \le (n_s + n_r \cdot q_{\mathrm{C}}) \cdot \mathsf{Adv}^{\mathsf{G}_5}_{\mathtt{Drivel}}(\mathcal{A}).$$

Let us write $\mathsf{G}_5^{\mathrm{T}}$ and $\mathsf{G}_5^{\mathrm{C}}$ for the game G_5 restricted to the two winning options of $\mathcal{A}$ ($\mathrm{A} \in \{\textsc{Test}, \textsc{ChallExec}\}$). By the union bound,

$$\mathsf{Adv}^{\mathsf{G}_5}_{\mathtt{Drivel}}(\mathcal{A}) \le \mathsf{Adv}^{\mathsf{G}_5^{\mathrm{T}}}_{\mathtt{Drivel}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{G}_5^{\mathrm{C}}}_{\mathtt{Drivel}}(\mathcal{A}).$$

We will bound each term separately, through the following proof cases I and II.

Case I. Win via Test

In this proof case, $\mathcal{A}$ commits to winning via a single Test query on a session π^*. For the test session to satisfy Fresh, we must have that there is an honest partner—i.e., a session π_p^* holding the same *cid* or *sid* ("passive execution")—or that the responder peer is uncompromised upon acceptance of the initiator session π^* ("forward secrecy"). Note that since $\mathsf{ExplicitAuth}$ is ensured to hold (by Game G_4), the latter implies that there actually must be an honest partner π_p^* holding the same *sid*, so we can at this point focus on such passive executions.

Game I.0. This case begins with Game G_5 conditioned on A = Test, where the test session has an honest (*sid* or *cid*) partner:

$$\mathsf{Adv}^{\mathsf{G}_{\mathrm{I}.0}}_{\mathtt{Drivel}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{G}_5^{\mathrm{T}}}_{\mathtt{Drivel}}(\mathcal{A}).$$

Game I.1 (Guess the partner session). In Game $\mathsf{G}_{\mathrm{I}.1}$, we guess the session π_p^* that is the honest *sid* partner (if π^* is an initiator) or *cid* partner (else) of π^*. Aborting if the guess is incorrect, this introduces a loss in the number of sessions:

$$\mathsf{Adv}^{\mathsf{G}_{\mathrm{I}.0}}_{\mathtt{Drivel}}(\mathcal{A}) = n_s \cdot \mathsf{Adv}^{\mathsf{G}_{\mathrm{I}.1}}_{\mathtt{Drivel}}(\mathcal{A}).$$

<u>Game I.2 (Ephemeral KEM IND-1CCA).</u> Let

$$(c_e, K_e) \leftarrow\!\!\$\ \mathsf{KEM.Encap}(pk_e)$$

be the encapsulation computed at session π^*. If π^* is an initiator then this is the ephemeral public key pk_e in *sid*, and otherwise, it is the ephemeral public key in *cid* (which must necessarily be determined by Game $\mathsf{G}_{\mathrm{I}.1}$).

In Game $\mathsf{G}_{\mathrm{I}.2}$, we replace the ephemeral KEM key K_e with a uniformly random $\widetilde{K}_e$ in π^*. All values derived from K_e in π^* use the randomized value $\widetilde{K}_e$.

We bound the adversary $\mathcal{A}$'s difference in advantage by a reduction $\mathcal{B}_8$ to the $\mathsf{IND\text{-}1CCA}$ security of KEM. $\mathcal{B}_8$ obtains the $\mathsf{IND\text{-}1CCA}$ challenge (pk, c^*, K^*) and simulates the game for $\mathcal{A}$ as follows. In the protocol run between π^* and π_p^*, $\mathcal{B}_8$ uses pk as the ephemeral public key of the initiator and c^* as the ciphertext c_e. If the initiator session receives a ciphertext $c_e \neq c^*$, then $\mathcal{B}_8$ queries its $\mathsf{IND\text{-}1CCA}$ decapsulation oracle (once) and uses the response as K_e; else, if $c_e = c^*$, then $\mathcal{B}_8$ uses K^* as K_e. If K^* is the real KEM key then $\mathcal{B}_8$ has exactly simulated $\mathsf{G}_{\mathrm{I}.1}$ to $\mathcal{A}$; else, if K^* is random, then $\mathcal{B}_8$ has exactly simulated $\mathsf{G}_{\mathrm{I}.2}$ to $\mathcal{A}$. Therefore:

$$\Pr[\mathsf{G}_{\mathrm{I}.1}] - \Pr[\mathsf{G}_{\mathrm{I}.2}] \leq \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{IND\text{-}1CCA}}(\mathcal{B}_8).$$

Game I.3 (Random *FS*). In Game $\mathsf{G}_{\mathrm{I}.3}$, we replace evaluations $\mathsf{F}_2(\cdot, K^*)$ with a random function. This in particular replaces FS with a uniformly random value $\widetilde{FS}$ in π^*. (Note that if π_p^* received the same ciphertext, then it will also use K^*.)

We bound the difference in this step by a reduction to the $\mathsf{swap\text{-}PRF}$ security of F_2. The reduction $\mathcal{B}_9$ uses its oracle in place of $\mathsf{F}_2(\cdot, K^*)$, simulating either $\mathsf{G}_{\mathrm{I}.2}$ or $\mathsf{G}_{\mathrm{I}.3}$, giving:

$$\Pr[\mathsf{G}_{\mathrm{I}.2}] - \Pr[\mathsf{G}_{\mathrm{I}.3}] \leq \mathsf{Adv}_{\mathsf{F}_2}^{\mathsf{swap\text{-}PRF}}(\mathcal{B}_9).$$

Game I.4 (Random *skey*). Finally, in Game $\mathsf{G}_{\mathrm{I}.4}$, we replace evaluations $\mathsf{F}_1(\widetilde{FS}, \cdot)$ with a random function. This in particular replaces $skey$ with a uniformly random value $skey^*$ in π^*. We again bound this game hop by a reduction to the PRF security of F_1:

$$\Pr[\mathsf{G}_{\mathrm{I}.3}] - \Pr[\mathsf{G}_{\mathrm{I}.4}] \leq \mathsf{Adv}_{\mathsf{F}_1}^{\mathsf{PRF}}(\mathcal{B}_{10}).$$

We now have in Game $\mathsf{G}_{\mathrm{I}.4}$ that the real and random session key output of the Test oracle are both randomly sampled. Also, any non-partnered session keys are independent of π^*, since the context input when deriving $skey$ is precisely the session identifier sid. Hence, $\mathcal{A}$ has no better chance than guessing the challenge bit b and so

$$\mathsf{Adv}_{\mathtt{Drivel}}^{\mathsf{G}_{\mathrm{I}.4}}(\mathcal{A}) = 0.$$

Case II. Win via ChallExec

In this proof case, $\mathcal{A}$ commits to making a single ChallExec query on the pth-created server, denoted v^* here (and not making any Test queries). We assume that $\mathcal{A}$ makes one ChallExec call, as otherwise its advantage is 0, and hence never calls RevSecretKey on p, as otherwise it would violate $\mathsf{ObfFresh}$ and lose.

Game II.0. This case begins with Game G_5 conditioned on $\mathrm{A} =$ ChallExec:

$$\Pr[\mathsf{G}_{\mathrm{II}.0}] = \Pr[\mathsf{G}_5^{\mathrm{C}}].$$

Game II.1 (Long-term KEM SPR-CCA). Let

$$(c_S, K_S) \leftarrow_{\$} \mathsf{OKEM.Encap}(pk_S)$$

be the encapsulation computed by the initiator when ChallExec is called.

In Game $\mathsf{G}_{\text{II}.1}$, we replace the ciphertext c_S, as well as the long-term KEM key K_S as follows. Instead of running $\mathsf{OKEM.Encap}$, the initiator samples $c_S \leftarrow\$ \{0,1\}^{\mathsf{cl}}$ and $K_S \leftarrow\$ \mathcal{K}$ uniformly at random.

We bound the adversary $\mathcal{A}$'s difference in advantage by a reduction $\mathcal{B}_{11}$ to the $\mathsf{SPR\text{-}CCA}$ security of OKEM. $\mathcal{B}_{11}$ obtains the $\mathsf{SPR\text{-}CCA}$ challenge (pk, c^*, K^*) with respect to a simulator $\mathcal{S}_\$$ that samples ciphertexts uniformly at random from $\{0,1\}^{\mathsf{cl}}$, and simulates the game for $\mathcal{A}$ as follows. It uses pk as the public key of the pth-created responder v^* committed to by the adversary. When ChallExec is called, $\mathcal{B}_{11}$ uses c^* as the ciphertext c_S. In any session of v^*, if the ciphertext c_S received is not c^*, then $\mathcal{B}_{11}$ queries its $\mathsf{SPR\text{-}CCA}$ decapsulation oracle and uses the response as K_S; else, if $c_S = c^*$, then $\mathcal{B}_{11}$ uses K^* as K_S. If (c^*, K^*) are the real KEM values then $\mathcal{B}_{11}$ exactly simulates $\mathsf{G}_{\text{II}.0}$ to $\mathcal{A}$; else, if they are random, then $\mathcal{B}_{11}$ simulates $\mathsf{G}_{\text{II}.1}$. Therefore:

$$\Pr[\mathsf{G}_{\text{II}.0}] - \Pr[\mathsf{G}_{\text{II}.1}] \le \mathsf{Adv}^{\mathsf{SPR\text{-}CCA}}_{\mathsf{OKEM},\mathcal{S}_\$}(\mathcal{B}_{11}).$$

Game II.2 (Random ES). In Game $\mathsf{G}_{\text{II}.2}$, we replace ES with a uniformly random $\widetilde{ES}$ in the sessions created by the ChallExec call. We bound the difference in this step by a reduction $\mathcal{B}_{12}$ to the $\mathsf{swap\text{-}PRF}$ security of F_2, using its oracle instead of computing $\mathsf{F}_2(\cdot, K_S)$. This yields

$$\Pr[\mathsf{G}_{\text{II}.1}] - \Pr[\mathsf{G}_{\text{II}.2}] \le \mathsf{Adv}^{\mathsf{swap\text{-}PRF}}_{\mathsf{F}_2}(\mathcal{B}_{12})$$

Game II.3 (Random ES', encryption keys, and MAC tags). Next, in Game $\mathsf{G}_{\text{II}.3}$, we replace any evaluations of F_1 keyed with $\widetilde{ES}$ with a random function in the sessions created by ChallExec. This in particular replaces ES', the encryption keys EK_1 and EK_2, and the MAC tags MAC_C, M_C, MAC_S, and M_S with uniformly random values $\widetilde{ES'}$, $\widetilde{EK}_1$, $\widetilde{EK}_2$, MAC^*_C, M^*_C, MAC^*_S, and M^*_S, respectively. We again bound the difference in this by a reduction to the PRF security of F_1:

$$\Pr[\mathsf{G}_{\text{II}.2}] - \Pr[\mathsf{G}_{\text{II}.3}] \le \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_1}(\mathcal{B}_{13}).$$

Game II.4 (Random FS). In Game $\mathsf{G}_{\text{II}.4}$, we replace FS with a uniformly random $\widetilde{FS}$ in the sessions created by the ChallExec call, again bounded by PRF security of F_2:

$$\Pr[\mathsf{G}_{\text{II}.3}] - \Pr[\mathsf{G}_{\text{II}.4}] \le \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_2}(\mathcal{B}_{14}).$$

Game II.5 (Random $skey$ and $auth$). In Game $\mathsf{G}_{\text{II}.5}$, we replace evaluations $\mathsf{F}_1(\widetilde{FS}, \cdot)$ with a random function in the sessions created by the ChallExec call. This in particular replaces $skey$ and $auth$ with uniformly random values $skey^*$ and $auth^*$, respectively. Bounding again by PRF security of F_1:

$$\Pr[\mathsf{G}_{\text{II}.4}] - \Pr[\mathsf{G}_{\text{II}.5}] \le \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}_1}(\mathcal{B}_{15}).$$

Note that at this point, in the ChallExec sessions, the ephemeral KEM key pair (sk_e, pk_e), the ephemeral KEM ciphertext c_e, and the shared secret K_e are not used anymore, since the values FS, $skey$, and $auth$ computed from them are sampled at random as per

Games $\mathsf{G}_{\text{II}.4}$ and $\mathsf{G}_{\text{II}.5}$. We will leverage this to replace the ephemeral ciphertext and public key with random strings in the transcript in the final game hops.

<u>Game II.6 (Random epk_e).</u> In Game $\mathsf{G}_{\text{II}.6}$, we replace $epk_e = \mathsf{SE.Enc}(\widetilde{EK}_1, pk_e)$ in the sessions created by the ChallExec call with a random bitstring epk_e^* of the same length. We can bound the adversary $\mathcal{A}$'s difference in advantage by a reduction $\mathcal{B}_{16}$ to the OT-IND\$ security of SE, since $\widetilde{EK}_1$ is random by $\mathsf{G}_{\text{II}.3}$. We obtain

$$\Pr[\mathsf{G}_{\text{II}.5}] - \Pr[\mathsf{G}_{\text{II}.6}] \leq \mathsf{Adv}_{\mathsf{SE}}^{\mathsf{OT\text{-}IND\$}}(\mathcal{B}_{16}).$$

<u>Game II.7 (Random ect_e).</u> Finally, Game $\mathsf{G}_{\text{II}.7}$ replaces $ect_e = \mathsf{SE.Enc}(\widetilde{EK}_2, c_e)$ in the sessions created by the ChallExec call with a random bitstring ect_e^* of the same length. Similar to $\mathsf{G}_{\text{II}.6}$, we bound the adversary $\mathcal{A}$'s difference in advantage by a reduction $\mathcal{B}_{17}$ to the OT-IND\$ security of SE, since $\widetilde{EK}_2$ is random by $\mathsf{G}_{\text{II}.3}$:

$$\Pr[\mathsf{G}_{\text{II}.6}] - \Pr[\mathsf{G}_{\text{II}.7}] \leq \mathsf{Adv}_{\mathsf{SE}}^{\mathsf{OT\text{-}IND\$}}(\mathcal{B}_{17}).$$

Now we have that the response to the ChallExec oracle is distributed independent of the challenge bit b: In both cases, the returned transcript consists of random strings of length corresponding to the protocol messages and variable padding, exactly matching the output of the simulator $\mathcal{S}_{\texttt{Drivel}}$ (Section 4.3): epk_e is random by $\mathsf{G}_{\text{II}.6}$, c_S by $\mathsf{G}_{\text{II}.1}$, ect_e by $\mathsf{G}_{\text{II}.7}$, P_C, P_S are random padding by definition, and M_C, MAC_C, M_S, MAC_S are replaced by uniformly random values (by $\mathsf{G}_{\text{II}.3}$), as well as the *auth* tag and the session key *skey* (by $\mathsf{G}_{\text{II}.5}$).

Hence $\mathcal{A}$ cannot win in this case anymore and we have

$$\mathsf{Adv}_{\texttt{Drivel}}^{\mathsf{G}_{\text{II}.7}}(\mathcal{A}) = 0.$$

Collecting the bounds yields the theorem statement. □