C'est très CHIC: A compact password-authenticated key exchange from lattice-based KEM

Afonso Arriaga[1], Manuel Barbosa[2], Stanislaw Jarecki[3], and Marjan Škrobot[1]

[1] SnT - University of Luxembourg
{afonso.delerue, marjan.skrobot}@uni.lu
[2] University of Porto (FCUP), INESC-TEC
and Max Planck Institute for Security and Privacy
mbb@fc.up.pt
[3] University of California at Irvine
stanislawjarecki@gmail.com

Abstract. Driven by the NIST's post-quantum standardization efforts and the selection of Kyber as a lattice-based Key-Encapsulation Mechanism (KEM), several Password Authenticated Key Exchange (PAKE) protocols have been recently proposed that leverage a KEM to create an efficient, easy-to-implement and secure PAKE. In two recent works, Beguinet et al. (ACNS 2023) and Pan and Zeng (ASIACRYPT 2023) proposed generic compilers that transform KEM into PAKE, relying on an Ideal Cipher (IC) defined over a group. However, although IC on a group is often used in cryptographic protocols, special care must be taken to instantiate such objects in practice, especially when a low-entropy key is used. To address this concern, Dos Santos et al. (EUROCRYPT 2023) proposed a relaxation of the IC model under the Universal Composability (UC) framework called Half-Ideal Cipher (HIC). They demonstrate how to construct a UC-secure PAKE protocol, EKE-KEM, from a KEM and a modified 2-round Feistel construction called m2F. Remarkably, the m2F sidesteps the use of an IC over a group, and instead employs an IC defined over a fixed-length bitstring domain, which is easier to instantiate.

In this paper, we introduce a novel PAKE protocol called CHIC that improves the communication and computation efficiency of EKE-KEM, by avoiding the HIC abstraction. Instead, we split the KEM public key in two parts and use the m2F directly, without further randomization. We provide a detailed proof of the security of CHIC and establish precise security requirements for the underlying KEM, including one-wayness and anonymity of ciphertexts, and uniformity of public keys. Our findings extend to general KEM-based EKE-style protocols and show that a passively secure KEM is not sufficient. In this respect, our results align with those of Pan and Zeng (ASIACRYPT 2023), but contradict the analyses of KEM-to-PAKE compilers by Beguinet et al. (ACNS 2023) and Dos Santos et al. (EUROCRYPT 2023).

Finally, we provide an implementation of CHIC, highlighting its minimal overhead compared to the underlying KEM – Kyber. An interesting aspect of the implementation is that we reuse the rejection sampling procedure in Kyber reference code to address the challenge of hashing onto the public key space. As of now, to the best of our knowledge, CHIC stands as the most efficient PAKE protocol from black-box KEM that offers rigorously proven UC security.

Keywords: Password Authenticated Key Exchange, Key Encapsulation Mechanism, Universal Composability, Post-Quantum, Ideal Cipher.

1 Introduction

The problem of attaining secure communication online is commonly addressed by employing Authenticated Key Exchange (AKE) protocols that involve high-entropy long-term private keys, often relying on Public Key Infrastructure (PKI). However, in scenarios where humans are involved in the authentication process, secure storage of long-term private keys by users is impractical, and most applications resort to a simpler and cost-effective solution—human-memorizable passwords. In most cases, applications carry out password-based authentication using (variants of) the bare-bones protocol where the user sends a password across the network to be checked with respect to a previously stored record (usually a salted hashed value) of the same password. This protocol, which is chosen due to its usability and ease of deployment, has a number of disadvantages from the security point of view. An obvious shortcoming is that the password is explicitly transferred across the communications channel, and so it requires a previously established secure and one-side-authenticated channel to the server checking the password. This opens the way to a number of well-known attacks, such as impersonating the server via a phishing attack.

Password Authenticated Key Exchange (PAKE) [6,5,9] is a cryptographic primitive that can mitigate some of the limitations associated with low-entropy passwords, and bootstrap a shared password into a cryptographically strong session key. Intuitively, PAKE protocols guarantee that the only way to extract a password from a user over the network is to actively perform a password-guessing attack by trying to run the protocol with the user multiple times.

The most efficient PAKE constructions to date, namely the CPACE protocol that has been recently chosen for standardization by the IETF [2], are built as variants of the Diffie-Hellman protocol and they achieve security with essentially no bandwidth overhead and minimal computational overhead—in CPACE this overhead is reduced to hash operations. Indeed, one of the takeaways of the CPACE selection process was that performance is critical for adoption.[4] This is because target applications include resource-constrained devices (e.g., IoT networks) and ad-hoc contexts (e.g., ePassports and file transfers). Therefore, a natural question to ask in the current context of migration to post-quantum

[4] `https://mailarchive.ietf.org/arch/msg/cfrg/usR4me-MKbW4QOOLprDKXu3TOHY`

secure cryptography is how to construct efficient PAKE protocols that are not Diffie-Hellman based and that, ideally, can leverage the recent results of the NIST post-quantum competition.

KEM-based PAKE protocols. In this direction, and very recently, several works [22,27,4,26,3] proposed black-box constructions of PAKE from a Key-Encapsulation Mechanism (KEM) and an Ideal Cipher (IC) or its variants (see below).[5] Conceptually, this KEM-based design paradigm sheds new light on the thirty-year-old *Encrypted Key Exchange* (EKE) approach to PAKE by Bellovin and Merritt [6]. From a practical point of view, this recent focus on the generic conversion of KEM into PAKE is largely driven by the efforts of the National Institute of Standards and Technology (NIST) to standardize Post-Quantum (PQ) cryptographic schemes, including KEM and digital signatures. In particular, the standardization of the first post-quantum KEM was just concluded [24] and the scheme is based on Crystals-Kyber, a module-lattice-based KEM. Kyber has undergone extensive scrutiny regarding its security and anonymity properties, as well as secure and efficient implementation, and this body of research can be leveraged when constructing PAKE protocols that use KEM in a black-box way.

A common characteristic of the above KEM-based PAKE proposals is their reliance on Random Oracle (RO) and Ideal Cipher (IC) models.[6] Despite the similarities among these proposed protocols, they still differ in subtle ways and can be categorized based on the model of analysis, design structure, and KEM security properties used to establish PAKE security. The protocols put forth by Bradley et al. [10], McQuoid et al. [22], Beguinet et al. [4] and Dos Santos et al. [27] are analysed under Universal Composability (UC) PAKE framework [12], while Pan and Zeng [26] and Alnahawi et al. [3] prove security under the game-based PAKE definition of Bellare-Pointcheval-Rogaway (BPR) [5]. We note that the UC PAKE security model of Canetti et al. [12] is significantly stronger than the BPR model. The superiority of the former springs fundamentally from the UC framework's ability to capture security under arbitrary correlations of password inputs, which is beyond the scope of current game-based PAKE security notions. Indeed, another important takeaway from the CPACE selection process within the IETF, was the relevance of a (thoroughly scrutinized) proof of security in the UC framework.[7]

Two approaches to KEM-based PAKE. Prior KEM-based PAKE protocols follow two distinct design patterns. Firstly, sPAKE [22], CAKE [4], and PAKE-KEM [26], follow a procedure where the initiator Alice employs an IC to encrypt a KEM public key under her password, and the responder Bob decrypts this public key and uses it to encapsulate a secret value.[8] This secret value

[5] This list can be extended by the PAPKE protocol of [10], which was originally presented as a generic PAKE from PKE and IC, but it can be recast as construction from KEM and IC.

[6] In [22] security is claimed based solely on RO, but that claim has not been formally established.

[7] https://mailarchive.ietf.org/arch/msg/cfrg/47pnOSsrVS8uozXbAuM-alEk0-s

[8] In sPAKE [22] the IC is replaced by a weaker primitive, see more below.

is used by both parties as an input to a hash function—modeled as a Random Oracle (RO)—to derive a session key. However, Bob does not send the KEM ciphertext to Alice in the clear but instead utilizes a second IC to encrypt the KEM ciphertext before transmitting it to the initiator. This approach ensures that both parties are committed to a single password via IC encryption, based on the collision-freeness of IC outputs. A practical disadvantage of this two-sided usage of IC is that it requires two distinct IC instances, one over the domain of KEM public keys, and the other over the domain of KEM ciphertexts. In lattice-based KEMs, these domains are typically different, and both of them are large, which makes implementing IC for these domains non-trivial.

The second design pattern, employed in protocols PAPKE [10], OCAKE [4], EKE-KEM [27], and PAKEM [3], takes a slightly different approach. Here, the KEM ciphertext obtained by Bob is sent in the clear, accompanied by a key confirmation *tag*, whose purpose is to make Bob's message a commitment to a single password guess.[9] The second design uses only one instance of IC, which makes it more efficient, and it does not require special properties of KEM ciphertexts, e.g. that they are indistinguishable from random elements of the ciphertext domain. In this work, we focus on efficiency and therefore adopt this design pattern.

Opening up the IC blackbox. The sPAKE and EKE-KEM protocols of resp. [22] and [27] deviate from the above pattern by replacing the Ideal Cipher on the domain of public keys (and ciphertexts in [22]) with a weaker and easier-to-construct primitive. One motivation for reducing the requirement on the password-based encryption component is the difficulty of efficiently instantiating IC on a group domain—cf. the discussion of the costs of possible approaches in e.g. [27], which is necessary to instantiate the "KEM+IC" design for PAKE using KEM instantiated as an Elliptic-Curve Diffie-Hellman. However, instantiating the same KEM+IC approach using a lattice-based KEM is also non-trivial because it would require a special-purpose IC on a domain of large bit strings (around one kilobyte in the case of Kyber). Even though there exist methods for extending an IC domain to bitstrings of arbitrary size, e.g., using Feistel networks, [13,16] these generic IC domain extension techniques would add significant complexity to an implementation and incur a significant performance penalty.

Motivated by the above, McQuoid et al. [22] proposed to replace IC in this KEM+IC approach to PAKE with a weaker primitive of a *Programmable-Once Public Function* (POPF), which they showed can be instantiated with a 2-round Feistel network (2F). In particular, in the case of Kyber KEM, the 2F encryption would involve just one RO hash onto the KEM public key domain, and one RO hash onto a domain of bitstrings of length 3λ, where λ is the security parameter.

[9] A seeming exception is the PAPKE protocol [10], which does not attach such a tag explicitly, but it requires a *strong robustness* property of the KEM, and the generic method for achieving this property includes expanding a CCA-secure KEM ciphertext with a key-committing tag [1]. Protocol PAKEM [3] also diverges from the pattern because it employs an additional message flow where Alice sends her own key confirmation tag to Bob. This last message achieves explicit mutual authentication in the Alice-to-Bob direction, but it adds an extra round to the protocol.

However, this way of implementing password encryption would add at least 384 (=3x128) bits to the KEM ciphertext. Moreover, as mentioned in footnote 6, the analysis of the resulting protocol as a UC-secure PAKE is currently incomplete. Dos Santos et al. [27] modify the 2-round Feistel network used by [22]—calling the result a *modified 2-Feistel* (m2F)—by reducing the bandwidth overhead to 256 (=2x128) bits: this is achieved at the cost of adding an IC on 256-bit strings into the encryption procedure. The security proof in [27] shows that m2F realizes a UC abstraction of a (randomized) *Half-Ideal Cipher* (HIC), and then shows that the above KEM+IC approach to UC PAKE works also in the case of KEM+HIC. However, because it is a randomized encryption using a 256-bit random seed, it adds at least 256 bits to the encrypted public key.

Main contribution: Compact m2F and bandwidth-minimal KEM-to-PAKE compiler. In this paper we revisit the construction of [27] and reduce the bandwidth overhead to a minimum. We observe that, for Kyber and other post-quantum KEMs, the public key can be split into two components, one of which is a 32-byte uniform seed ρ, and ask the following natural question:

> *Can we reduce the bandwidth overhead of the* m2F *by using ρ as the ephemeral randomness r in the* m2F *construction?*

We answer this question in the affirmative by giving direct proof that the resulting construction is a UC secure PAKE in the joint Ideal Cipher and Random Oracle model. By *direct proof* we mean that we do not rely on the Half-Ideal-Cipher abstraction of [27], and instead perform the proof over the fully expanded construction. The reason for this is that the notion of a UC-secure Half-Ideal-Cipher crucially relies on the fact that the m2F construction is randomized, i.e., that honest parties choose an ephemeral randomness that is *independent* of the input public key. By unifying this ephemeral randomness with a public-key component we lose this property and the ability to modularize the m2F construction. We call our construction CHIC for Compact Half-Ideal-Cipher, as a way to acknowledge the inspiration in the work of [27].

Second contribution: Requirements for the KEM. We provide a detailed proof of the security of CHIC and establish precise security requirements for the underlying KEM, including passive one-way security (OW-CPA) and pseudo-uniformity of public-keys (UNI-PK), necessary to achieve UC PAKE security. Like prior works, our proof shows that anonymity is also a necessary property for the security of the construction. However, we show that passive anonymity (i.e., indistinguishability of public keys and ciphertexts) is *not* sufficient to conclude the proof. We show that CHIC requires a ANO-1PCA-secure KEM[10], and that our analysis extends to the proofs by Beguinet et al. [4] and Dos Santos et al. [27], despite the claims that ANO-CPA-secure KEM would be sufficient.

Practical contribution: Implementation and experimental evaluation. We give an implementation of the protocol, clarifying all aspects of real-world deployment of the protocol, and we confirm experimentally the efficiency properties

[10] We refer the reader to Definition 3.

of the protocol. Our implementation builds on the reference implementation of Kyber—the full construction offering CCA security [8,28] and anonymity [15,20,29]. We clarify how to instantiate the m2F components showing, in particular, that hashing into the public-key space of Kyber can be done by reusing the code that the Kyber created for expanding the seed ρ in the public key to a matrix over the algebraic ring that underlies the KEM construction. Technically, this entails proving that the rejection sampling procedure specified by Kyber is indifferentiable from a random oracle; a result that may be of independent interest. Compared to EKE-KEM [27], CHIC saves 32 bytes in bandwidth costs, while also bringing mild computational savings by (1) eliminating the need for Alice to generate 32 bytes of random coins, and (2) simplifying the inputs/outputs of the m2F construction, with the right wire carrying only part of the public key. The implementation is available as supplementary material.

2 Preliminaries

In this section, we present the definition of Key Encapsulation Mechanism (KEM) and introduce its security properties of interest for this work.

Definition 1. *A Key Encapsulation Mechanism* (KEM) *scheme is a tuple of polynomial-time algorithms* KEM = (Keygen, Encap, Decap) *that behaves as follows:*

- Keygen(λ) $\rightarrow$ (pk, sk)*: a key-generation algorithm that on input a security parameter* λ*, outputs a public/private key pair* (pk, sk)*.*
- Encap(pk) $\rightarrow$ (c, K)*: an encapsulation algorithm that on input a public key* pk*, generates a ciphertext* c *and a secret key* K*.*
- Decap(sk, c) $\rightarrow$ K*: a decapsulation algorithm that on input a private key* sk *and a ciphertext* c*, output a secret key* K*.*

For correctness, we require that for any key pair $(pk, sk) \leftarrow$ Keygen(λ)*, and ciphertext and secret key* $(c, K) \leftarrow$ Encap(pk)*, we have that* $K =$ Decap(sk, c)*.*

KEM security properties. The standard security notion for key encapsulation mechanisms and public-key encryption in general is indistinguishability under chosen-ciphertext attacks (IND-CCA). In addition to achieving IND-CCA security, many applications also demand the property of anonymity in a KEM. An anonymous KEM ensures that a ciphertext conceals the identity of the recipient by revealing no information about the public key employed in the encapsulation process. Unsurprisingly, the standard definition of this property is a logical adaptation of IND-CCA known as 'anonymity under chosen-ciphertext attacks' (ANO-CCA), and with the adversary's objective being to determine which of two public keys was used to generate the given challenge ciphertext.

It has been widely established that Kyber is IND-CCA [8,28] and ANO-CCA [15,20,29]. However, for our construction, we rely on weaker properties, namely ‘one-wayness under plaintext-checkable attacks’ (OW-PCA)[11] and anonymity under plaintext-checkable attacks (ANO-PCA). In addition, we require a KEM with ‘splittable and pseudo-uniform public-keys’ (UNI-PK). All these notions are defined in Fig. 1.

```
Exp $\mathsf{ANO\text{-}iPCA}^{\mathcal{A}}_{\mathsf{KEM}}(\lambda)$
$(pk_0, sk_0) \leftarrow \mathsf{Keygen}(\lambda)$
$(pk_1, sk_1) \leftarrow \mathsf{Keygen}(\lambda)$
$b \leftarrow\!\!\$\ \{0,1\}$
$(c^*, \_) \leftarrow \mathsf{Encap}(pk_b)$
$b' \leftarrow \mathcal{A}^{\mathsf{PCO}_{i,c^*}(\mathsf{sk}_0,\cdot,\cdot)}(pk_0, pk_1, c^*)$
$\mathbf{return}\ b == b'$

Oracle $\mathsf{PCO}_{i,c^*}(\mathsf{sk}, c, K)$
$[\mathcal{A}$ can only make $i$ queries$]$
if $c == c^* \mathbf{return}\ \bot$
$\mathbf{return}\ K == \mathsf{Decap}(\mathsf{sk}, c)$

Exp $\mathsf{OW\text{-}iPCA}^{\mathcal{A}}_{\mathsf{KEM}}(\lambda)$
$(pk, sk) \leftarrow \mathsf{Keygen}(\lambda)$
$(c^*, \_) \leftarrow \mathsf{Encap}(pk)$
$K \leftarrow \mathcal{A}^{\mathsf{PCO}_{i,\bot}(\mathsf{sk},\cdot,\cdot)}(pk, c^*)$
$\mathbf{return}\ K == \mathsf{Decap}(sk, c^*)$

Exp $\mathsf{UNI\text{-}PK}^{\mathcal{A}}_{\mathsf{KEM},\mathsf{Split}}(\lambda)$
$(pk_0, \_) \leftarrow \mathsf{Keygen}(\lambda)$
$(r_0, M_0) \leftarrow \mathsf{Split}(pk_0)$
$(r_1, M_1) \leftarrow N_\lambda \times G_\lambda$
$b \leftarrow\!\!\$\ \{0,1\}$
$b' \leftarrow \mathcal{A}(r_b, M_b)$
$\mathbf{return}\ b == b'$
```

Fig. 1. Security experiments defining properties of KEM: (1) One-Wayness under Plaintext-Checkable Attacks (OW-iPCA); (2) Anonymity under Plaintext-Checkable Attacks (ANO-iPCA); (3) Splittable and pseudo-Uniform Public-Keys (UNI-PK). $\mathcal{A}$ is restricted to making at most i queries to the plaintext-checking oracle PCO. In particular, if $i = 0$, the plaintext-checking oracle PCO is not available. In ANO-iPCA security experiment, $\mathcal{A}$ is not allowed to query the plaintext-checking oracle PCO on the challenge ciphertext c^*. This restriction is *not* imposed in OW-iPCA experiment.

To elaborate, we first adopt the notion of one-wayness under plaintext checkable attacks from [25]. We consider an adversary whose goal is to decrypt a KEM ciphertext without the private decapsulation key but with access to a plaintext-checking oracle. This oracle allows the adversary to confirm if the decapsulation of a ciphertext under the challenge decryption key corresponds to a particular plaintext (i.e., the secret key K in the context of KEM).[12]

[11] Our construction can be proven secure assuming the underlying KEM has only ‘one-wayness under chosen-plaintext attacks’ (OW-CPA), though the proof is less tight.

[12] In IND-CCA security game, the decapsulation oracle can be queried on anything except the challenge ciphertext. However, in OW-PCA, the plaintext-checking oracle is unrestricted. Despite this gap, one can show that IND-CCA implies OW-PCA with a tight reduction, losing only a constant factor 2. For the proof, we refer to Appendix B.

Definition 2. ***(KEM one-wayness under plaintext-checkable attacks)*** *A Key Encapsulation Mechanism (KEM) scheme is said to be* $\mathsf{OW\text{-}iPCA}$ *secure if for any PPT adversary* $\mathcal{A}$ *engaged in the* $\mathsf{OW\text{-}iPCA}$ *security game, where* $\mathcal{A}$ *is restricted to making at most* i *queries to the plaintext-checking oracle* PCO*, the advantage of* $\mathcal{A}$ *defined as:*

$$\mathsf{Adv}^{\mathsf{OW\text{-}iPCA}}_{\mathsf{KEM},\mathcal{A}}(\lambda) \overset{\text{def}}{=} \Pr[\mathsf{OW\text{-}iPCA}^{\mathcal{A}}_{\mathsf{KEM}}(\lambda) = 1] \tag{1}$$

is a negligible function of the security parameter λ*. Experiment* $\mathsf{OW\text{-}iPCA}$ *is defined in Fig. 1.*

Similarly, we also adopt a weaker variant of the $\mathsf{ANO\text{-}CCA}$ property, where the decapsulation oracle is replaced with the less-capable plaintext-checking oracle. We call this definition $\mathsf{ANO\text{-}PCA}$. In $\mathsf{ANO\text{-}CCA}$ game, the decryption oracle disallows queries on the challenge ciphertext. We preserve this restriction in $\mathsf{ANO\text{-}PCA}$ so it is trivial to see that $\mathsf{ANO\text{-}CCA}$ implies $\mathsf{ANO\text{-}PCA}$ with no tightness loss in the reduction, as the plaintext-checking oracle could be easily simulated with a decapsulation oracle.

Definition 3. ***(KEM anonymity under plaintext-checkable attacks)*** *A Key Encapsulation Mechanism (KEM) scheme is said to be* $\mathsf{ANO\text{-}iPCA}$ *secure if for any PPT adversary* $\mathcal{A}$ *engaged in the* $\mathsf{ANO\text{-}iPCA}$ *security game, where* $\mathcal{A}$ *is restricted to making at most* i *queries to the plaintext-checking oracle* PCO *and is prohibited from calling the oracle on the challenge ciphertext, the advantage of* $\mathcal{A}$ *defined as:*

$$\mathsf{Adv}^{\mathsf{ANO\text{-}iPCA}}_{\mathsf{KEM},\mathcal{A}}(\lambda) \overset{\text{def}}{=} 2 \cdot \Pr[\mathsf{ANO\text{-}iPCA}^{\mathcal{A}}_{\mathsf{KEM}}(\lambda) = 1] - 1 \tag{2}$$

is a negligible function of the security parameter λ*. Experiment* $\mathsf{ANO\text{-}iPCA}$ *is defined in Fig. 1.*

Finally, a less common security requirement but which proved to be essential for the constructions of PAKE protocol from KEM and IC [27,4,26,3] is public key indistinguishability from uniform. In other words, the public keys output by the KEM key generation algorithm must be computationally indistinguishable from public keys uniformly sampled from the same key space. This notion is also known as fuzziness [4,26]. In this work, we extended the requirements for KEM public keys. Namely, CHIC requires a KEM with splittable and uniform public keys that meet the following criteria: (1) the public key can be encoded as a bitstring and a group element; (2) a random oracle indifferentiable hash onto the group exists; and (3) honestly generated and decomposed public keys appear uniformly distributed.[13]

[13] Later in our construction, the KEM public key will be split into two parts. The first part will be expanded to match the range of the second part—which is an element of a group—and used to mask the second part of the public key. The expansion function will be treated as a random oracle in our security proof. While it

Definition 4. ***(KEM with splittable and pseudo-uniform public keys)*** *A KEM scheme has splittable and pseudo-uniform public keys if (1) there exists an efficiently computable and invertible map* $\mathsf{Split} : \mathcal{PK}_\lambda \to N_\lambda \times G_\lambda$*, such that each security parameter* λ *defines domains* $\mathcal{PK}_\lambda$*,* G_λ*, and* $N_\lambda = \{0,1\}^{p(\lambda)}$ *for some polynomial p; (2) there exists an RO-indifferentiable hash from* N_λ *onto* G_λ*; (3) for any PPT adversary* $\mathcal{A}$ *engaged in the* UNI-PK *security game, the advantage of* $\mathcal{A}$ *defined as:*

$$\mathsf{Adv}^{\mathsf{UNI\text{-}PK}}_{\mathsf{KEM},\mathsf{Split},\mathcal{A}}(\lambda) \overset{\text{def}}{=} 2 \cdot \Pr[\mathsf{UNI\text{-}PK}^{\mathcal{A}}_{\mathsf{KEM},\mathsf{Split}}(\lambda) = 1] - 1 \tag{3}$$

is a negligible function of the security parameter λ*. Experiment* UNI-PK *is defined in Fig. 1.*

PQ KEMs with spittable and uniform public keys. Crystals-Kyber [28] public key consists of a seed $\rho \in \{0,1\}^{256}$ and a group element $\mathbf{t} \in R_q^k$, where R_q is the ring $\mathbb{Z}_q[X]/(X^n+1)$, $q = 3329$ is a small prime, $n = 256$ and $k \in \{2,3,4\}$ depending on the choice of the security parameter λ. Consider the split algorithm for Kyber KEM to be the trivial breakdown of Kyber public keys into these two components. Seed ρ is derived from expanding a purely random bitstring $d \in \mathbb{B}^{32}$ using a hash function $G(d)$ that produces two 32-byte outputs, with ρ being one of them. In the security proofs of Kyber [28,15,20,29], function G is modeled as a random oracle, which ensures that the distribution of ρ is uniform. In FIPS 203 [24] standard, function G is specified to be instantiated as SHA3-512. ρ is then further expanded into a large stream of candidate 12-bit values via an eXtendable Output Function (XOF), which Kyber instantiates with SHAKE-128. From this stream, the first candidate values within the range $[0,3329)$ are selected to form the public matrix $\mathbf{A} \in R_q^{k\times k}$ in the NTT domain. This process is known as *rejection sampling* and, again, can be modelled as a random oracle mapping ρ to $\mathbf{A}$. Matrix $\mathbf{A}$ is then used to compute the second component of the public key as a Module-LWE instance. Therefore, the Kyber public key can be shown to be pseudo-uniform under the decisional MLWE assumption [4] in the Random Oracle Model. To instantiate the RO that maps elements in $N_\lambda = \{0,1\}^{256}$ to $G_\lambda = R_q^k$, we borrow the same rejection sampling procedure from Kyber. We expand this intuition and show that Kyber has splittable and pseudo-uniform public keys in Appendix C.

Other lattice-based KEMs employ the same technique of expanding a short seed into a public matrix, making them good candidates for splittable keys. For

is generally accepted that random oracles with fixed ranges can be easily instantiated with cryptographically-secure hash functions, the instantiation of a random oracle for hashing into the group where (part of) the KEM public keys reside is less straightforward. Indifferentiability [21] allows to formally justify the instantiation of a non-trivial hashing procedure: it ensures that one can safely replace an ideal object (e.g., a RO that hashes into a group) with a construction that makes use of another ideal object (e.g., an ideal eXtendable Output Function). The requirement (2) emphasizes the need for such a hashing procedure to safely instantiate our protocol.

example, FrodoKEM [23], the lattice-based KEM recommended by the German Federal Office for Information Security (BSI)[14], also has splittable and pseudo-uniform public keys. Similarly to Kyber, FrodoKEM public keys can be trivially decomposed into a seed $seed_A \in \{0,1\}^{128}$ (that expands to an $n \times n$ matrix $\mathbf{A}$, where all the coefficients are in $\mathbb{Z}_q$), and a group element $\mathbf{B} \in \mathbb{Z}_q^{n \times 8}$. The instantiation of the RO-indifferentiable hash-onto-group is even simpler for FrodoKEM because q is required to be a power of 2, so rejection sampling is not needed.

CPA versus iPCA. Some essential points should be noted concerning these security definitions. Firstly, when access to the PCO oracle is restricted to zero queries, it effectively results in the removal of the oracle from the experiment. This, in turn, gives rise to the weaker definitional variants known as 'chosen-plaintext attacks,' specifically OW-CPA and ANO-CPA. Furthermore, we made two adjustments to weaken our ANO-iPCA definition: (a) we refrained from providing the adversary with the challenge secret key K^*, and (b) we restricted the PCO oracle to queries on the left private decapsulation key sk_0. This contrasts with definitions in [15,20,29], which grant the adversary access to both keys via the oracle. These adaptations, which relax the requirements of the underlying KEM, are proven to be sufficient for establishing the security of the protocol CHIC presented in this paper.

It is also worth mentioning that for a very limited number of queries to the PCO oracle, OW-iPCA is equivalent to OW-CPA, as established by Lemma 1. However, it is essential to recognize that this equivalence cannot be readily extended to indistinguishability-based games. In such games, a flawed simulation resulting from an incorrect coin flip could nullify the advantage gained when the simulation was correct. Consequently, we cannot make a similar assertion regarding the relationship between ANO-iPCA and ANO-CPA.

Lemma 1. *If* KEM *is a* OW-CPA *secure key encapsulation mechanism, then it is also* OW-1PCA *secure.*

Proof. Let $\mathcal{A}$ be any adversary against game OW-1PCA. We construct an adversary $\mathcal{B}$ against OW-CPA that simulates game OW-1PCA for $\mathcal{A}$ as follows: i. Challenge (pk, c^*) is forwarded to $\mathcal{A}$. ii. The single oracle query to PCO is answered by $\mathcal{B}$ with a coin flip. iii. Finally, $\mathcal{B}$ forwards $\mathcal{A}$'s answer to OW-1PCA as its own answer to OW-CPA.

Notice that $\mathcal{B}$ perfectly simulates OW-1PCA for $\mathcal{A}$ half of the time, no matter what is $\mathcal{A}$'s strategy for querying the plaintext-checking oracle. Therefore, at least half of the time (possibly more, in case $\mathcal{A}$ wins regardless of the bad simulation of PCO), a win for $\mathcal{A}$ translates into a win for $\mathcal{B}$.

$$\mathsf{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{KEM},\mathcal{B}}(\lambda) \;\geq\; \frac{1}{2} \cdot \mathsf{Adv}^{\mathsf{OW\text{-}1PCA}}_{\mathsf{KEM},\mathcal{A}}(\lambda) \tag{4}$$

In broader terms, OW-iPCA is essentially equivalent to OW-CPA, but only when the number of queries made to the plaintext-checking oracle is limited to

[14] BSI TR-02102-1, Version: 2024-1

a few, as attempting to guess the PCO oracle's responses multiple times leads to an exponential loss in the number of tosses.

Definition 5. ***(Modified 2-Feistel construction: m2F)*** *The modified 2-round Feistel network, as introduced in [27], is constructed using three components: (1) block cipher denoted by the tuple of algorithms* $(\mathsf{IC.Enc}, \mathsf{IC.Dec})$*, with key space* K *and input/output space* N*; (2) hash function* H *whose output space is represented by group* G*; and (3) hash function* H' *whose output space is* K*. The* $\mathsf{m2F}$ *construction encompasses two efficiently computable functions,* $\mathsf{m2F}_{pw} : N \times G \to N \times G$ *and its inverse* $\mathsf{m2F}_{pw}^{-1}$*, both shown in Figure 2.*

```
$\mathsf{m2F}_{pw}(r, M)$
    $R \leftarrow \mathsf{H}(pw, r)$
    $T \leftarrow M \odot R$
    $t \leftarrow \mathsf{H}'(pw, T)$
    $s \leftarrow \mathsf{IC.Enc}(t, r)$
    **return** $(s, T)$

$\mathsf{m2F}_{pw}^{-1}(s, T)$
    $t \leftarrow \mathsf{H}'(pw, T)$
    $r \leftarrow \mathsf{IC.Dec}(t, s)$
    $R \leftarrow \mathsf{H}(pw, r)$
    $M \leftarrow T \odot R^{-1}$
    **return** $(r, M)$
```

Fig. 2. The modified 2-Feistel [27], where $\odot$ is a group G operation, and $(\cdot)^{-1}$ is an inverse in G.

Half-Ideal Cipher (HIC). In [27], the authors introduce a UC security notion they called (randomized) *Half-Ideal Cipher* (HIC), which is designed to relax the UC notion of an ideal cipher. This security notion is established through the introduction of an ideal functionality, denoted as $\mathcal{F}_{\mathsf{HIC}}$, and is parameterized by the domain $N \times G$. Notably, $\mathcal{F}_{\mathsf{HIC}}$ features 'honest' interfaces accessible for queries by the environment $\mathcal{Z}$, with these queries being mediated through honest parties. However, the honest interfaces are restricted w.r.t. to half of the input: $\mathcal{Z}$ has no control over the randomness parameter $r \in N$ in the encryption direction, and it cannot observe the value of r during decryption. By contrast, $\mathcal{F}_{\mathsf{HIC}}$ provides two adversarial interfaces that grant the adversary/simulator the capability to select r and even program half of the output $T \in G$ during encryption. In the decryption direction, the adversary can also observe the value of r.

It is shown in [27] that the $\mathsf{m2F}$ construction realizes $\mathcal{F}_{\mathsf{HIC}}$ functionality in the Random Oracle and Ideal Cipher (IC) model. The HIC abstraction serves as an effective replacement for an ideal cipher in the construction of EKE-like protocols, eliminating the need for the direct use of an IC over groups, whose instantiations are non-trivial (e.g., see [27]). However, it's worth noting that the randomized encryption of HIC introduces an overhead equal to the length of r. Due to the security proof of $\mathsf{m2F}$ requiring no collisions on the domain of the IC, this overhead essentially amounts to 2λ bits, which is precisely what our construction CHIC optimizes.

HIC+ and why it fails. A natural question is whether the $\mathcal{F}_{\mathsf{HIC}}$ could be extended to $\mathcal{F}_{\mathsf{HIC}+}$ which empowers honest parties and provides them with the ability to select and have visibility over r in respectively encryption and decryption. Unfortunately, the $\mathsf{m2F}$ construction would not be a provably secure realization of such extended functionality. To see why, let us exemplify with a concrete attack coordinated between an environment $\mathcal{Z}$ and its adversary $\mathcal{A}$:

1. $\mathcal{Z}$ selects r and M at random from the respective domains, picks arbitary pw, queries $\mathcal{F}_{\mathsf{HIC}+}$, via a honest party, on $\mathsf{Enc}(pw,(r,M))$, and obtains some ciphertext (s,T).
2. $\mathcal{Z}$ queries $\mathsf{H}'(pw,T)$ via its adversary $\mathcal{A}$ and obtains t (a key for IC).
3. $\mathcal{Z}$ queries $\mathsf{IC.Enc}(t,r)$ via its adversary $\mathcal{A}$ and should get back s.

Unfortunately, the simulator SIM cannot possibly know how to correctly answer the last query because it has no visibility over the first query $\mathcal{Z}$ made to $\mathcal{F}_{\mathsf{HIC}+}$, even though it controls IC, H and H'. This would not happen using $\mathcal{F}_{\mathsf{HIC}}$ interfaces because the environment $\mathcal{Z}$ can only pass message M to the honest party Enc interface in step (1) above, and it would not know the randomness r (which in the real-world would be internally chosen by that honest party).

Although we could not leverage the modular abstraction that $\mathcal{F}_{\mathsf{HIC}}$ introduces (or an extension of it), we still take full advantage of the $\mathsf{m2F}$ construction, as a white-box drop-in, in our protocol CHIC and rely directly on the RO and IC in the security proof. Therefore, no security definition is formally introduced here for $\mathcal{F}_{\mathsf{HIC}}$, and $\mathsf{m2F}$ is not explicitly parameterized by a security parameter λ, although its internal components are essential to the security analysis of CHIC.

3 Security Model

We begin this section with a brief review of the Universal Composability (UC) framework. Then we present the standard PAKE functionality as defined by Canetti et al. [12].

Let $\mathcal{P}$ be a protocol of interest whose security properties are modelled within the UC framework. In this framework, the environment $\mathcal{Z}$ embodies some higher-level protocol that uses $\mathcal{P}$ as a sub-protocol, while also acting as an adversary attacking that higher-level protocol. Here, the adversary $\mathcal{A}$ represents the adversary attacking protocol $\mathcal{P}$. Between the environment $\mathcal{Z}$ and the adversary there is a continuously open communication channel. Such setup allows $\mathcal{Z}$ to launch an attack on the higher-level protocol with the help of $\mathcal{A}$ (who is attacking protocol $\mathcal{P}$). Note that $\mathcal{Z}$ can only indirectly (through adversary $\mathcal{A}$) make calls to idealized primitives such as an Ideal Cipher and/or a Random Oracle.

In the UC framework that models the security of PAKE protocols, parties are initialized by the environment $\mathcal{Z}$ with arbitrary passwords of the environment's choice. In the real world, protocols are executed according to protocol specifications, in the presence of an adversary $\mathcal{A}$ capable of dropping, injecting, and modifying protocol messages at will, thus modelling an insecure network. In the ideal world, parties do not execute the protocol. Instead, they interact via an ideal functionality $\mathcal{F}_{\mathsf{PAKE}}$ described in Figure 3, in the presence of a simulator SIM that acts as an adversary operating in the ideal world. The simulator SIM is also allowed to interact with $\mathcal{F}_{\mathsf{PAKE}}$, but only using the $\mathcal{F}_{\mathsf{PAKE}}$ adversarial interfaces as defined in Fig. 3.

New Session. On $(\mathsf{NewSession}, sid, P_i, P_j, pw_C, role)$ from party $\mathcal{P}_i$:

- Ignore this query if two or more records of the form $(sid, ...)$ already exist.
- Else record $(sid, P_i, P_j, \mathsf{fresh}, pw_C, \bot)$ and send $(\mathsf{NewSession}, sid, P_i, P_j, role)$ to $\mathcal{A}$.

Test Password Guess. On $(\mathsf{TestPw}, sid, P_i, pw^*)$ from adversary $\mathcal{A}$:

- Retrieve record $(sid, P_i, P_j, \mathsf{fresh}, pw_C, \bot)$, abort if no such record exists.
- If $pw^* = pw_C$, then update the record to $(sid, P_i, P_j, \mathsf{compromised}, pw_C, \bot)$ and send $(\mathsf{TestPw}, sid, \mathsf{correct})$ to $\mathcal{A}$.
- Else update record to $(sid, P_i, P_j, \mathsf{interrupted}, pw_C, \bot)$ and send $(\mathsf{TestPw}, sid, \mathsf{wrong})$ to $\mathcal{A}$.

Session Key. On $(\mathsf{NewKey}, sid, P_i, k^*)$ from adversary $\mathcal{A}$ where $|k^*| = \lambda$:

- Retrieve record $(sid, P_i, P_j, status, pw, \bot)$ for $status \in \{\mathsf{fresh}, \mathsf{interrupted}, \mathsf{compromised}\}$, abort if no such record exist.
- If $status = \mathsf{compromised}$, set $k \leftarrow k^*$.
- If $status = \mathsf{fresh}$ and there exists a record $(sid, P_j, P_i, \mathsf{completed}, pw, k')$ whose $status$ switched from fresh to $\mathsf{completed}$ when P_j received $(\mathsf{NewKey}, sid, k')$, set $k \leftarrow k'$.
- Else set $k \leftarrow\$ \{0,1\}^\lambda$.
- Update the record to $(sid, P_i, P_j, \mathsf{completed}, pw, k)$, and output $(\mathsf{NewKey}, sid, k)$ to P_i.

Fig. 3. The PAKE ideal functionality $\mathcal{F}_{\mathsf{PAKE}}$ of Canetti et al. [12].

Finally, the goal of the environment $\mathcal{Z}$ that interacts with the parties and the adversary (either real world $\mathcal{A}$ or ideal world SIM) is to guess if it is in the real or in a simulation of the ideal world. Consequently, if for every efficient adversary $\mathcal{A}$ no such efficient environment $\mathcal{Z}$ exists that distinguishes the real world from the ideal world, we say that the protocol of interest $\mathcal{P}$ securely emulates ideal functionality $\mathcal{F}_{\mathsf{PAKE}}$. The UC PAKE definition results in a stronger notion than game-based PAKE notions and successfully captures the scenario where clients register related passwords with different servers, as this is captured by the ability of $\mathcal{Z}$ initializing parties with passwords of its choosing. Furthermore, the UC framework also ensures security under arbitrary protocol composition. Note that the environment $\mathcal{Z}$ may reveal various information to the adversary $\mathcal{A}$, thus allowing UC PAKE definitions to capture password leaks (static adversaries) and internal state leaks (adaptive adversaries) that may occur anytime during the protocol execution.

4 UC PAKE from Modified 2-Feistel and KEM

In this section, we present CHIC, a UC-secure Password Authenticated Key Exchange protocol. CHIC assumes a KEM scheme that is one-way secure (at least $\mathsf{OW\text{-}CPA}$, but $\mathsf{OW\text{-}PCA}$ results in a tighter proof), anonymous ($\mathsf{ANO\text{-}1PCA}$), and has splittable and pseudo-uniform public keys ($\mathsf{UNI\text{-}PK}$). The protocol, shown in Figure 4, is built upon the modified 2-Feistel ($\mathsf{m2F}$) construction of Dos Santos et al. [27]. We take a moment to discuss several design choices in our protocol, which follows an EKE-style construction combining a KEM and the $\mathsf{m2F}$.

First, a pivotal decision, in contrast to the strategy in [27], was to 'derandomize' the $\mathsf{m2F}$. We split the KEM public key and use the parts as inputs to the $\mathsf{m2F}$. This approach helps us avoid employing an ideal cipher over a group, which can be both costly and challenging to instantiate and eliminates any communication overhead associated with the HIC abstraction.

Second, the inputs used for tag and session key generation realized through the H_1 and H_2 function calls in CHIC, are identical. This allows us to optimize our implementation by making a single call to a hash function with an extended output size of 2λ. Subsequently, the output is cut in two halves, one forming the tag and the other the session key.

Third, the password is exclusively used in the $\mathsf{m2F}$ construction and is not provided as input to either H_1 or H_2. This design choice means that the initiator does not need to store the input password in memory while waiting for the responder's answer. This choice has potential benefits in the event of a complete compromise of the initiator (including leakage of its internal state), as an attacker would be required to perform an offline dictionary attack to retrieve the initiator's password under such circumstances. (However, we don't analyse the security of our protocol under adaptive attacks in the UC sense, and this sort of attack scenario is not captured by the $\mathcal{F}_{\mathsf{PAKE}}$ functionality.)

Fourth, it is worth noting that in our protocol the initiator, instead of aborting, outputs a random session key in the event that the received tag is invalid. We opt for this approach to ensure that our construction aligns with the security requirements specified in the standard UC PAKE functionality from [12] that foresees implicit authentication. However, in practice, when implementing the protocol, it is possible for the initiator to abort in that case, thus achieving explicit responder-to-initiator authentication. Furthermore, it is assumed that protocol participants erase any internal state as soon as it becomes unnecessary for the execution of the protocol. This means that the initiator instance after computing and sending apk erases its entire internal state (including the password) except $fullsid$, apk, pk, and sk.

Note that if function Split can be randomized, specifically if $\mathsf{Split}(pk)$ returns (r, pk) for $r \leftarrow\!\!\$\ N$ and $\mathsf{Split}^{-1}(r, pk)$ returns pk, then the $\mathsf{Split}{+}\mathsf{m2F}$ block in protocol CHIC would instantiate the randomized Half-Ideal Cipher construction of [27]. In that sense, the $\mathsf{Split}{+}\mathsf{m2F}$ procedure used in CHIC can be seen as a strict generalization of the HIC construction of [27].

5 Security Analysis

In this section, we prove that the protocol described in Figure 4 UC-realizes the standard PAKE functionality $\mathcal{F}_{\mathsf{PAKE}}$ shown in Figure 3.

Theorem 1. *Let* KEM *be a* $\mathsf{OW\text{-}CPA}$*,* $\mathsf{ANO\text{-}1PCA}$*, and* $\mathsf{UNI\text{-}PK}$*-secure key encapsulation mechanism. Let* IC *be a block cipher modeled as an ideal cipher, and* H*,* H'*,* H_1 *and* H_2 *be hash functions modeled as random oracles. Then, the PAKE protocol* CHIC *described in Fig. 4 UC-realizes* $\mathcal{F}_{\mathsf{PAKE}}$ *in the static corruption model. Furthermore, a* $\mathsf{OW\text{-}PCA}$*-secure* KEM *leads to a tighter proof.*

Proof overview. To prove Theorem 1 we show that the environment cannot distinguish between the "real world" experiment in which the environment $\mathcal{Z}$ and adversary $\mathcal{A}$ have parties P_i and P_j execute the protocol from Fig. 4, from an "ideal world" experiment in which a simulator SIM interacts with $\mathcal{F}_{\mathsf{PAKE}}$ and presents to environment $\mathcal{Z}$ a view that is consistent with what $\mathcal{A}$ produces in the real world. We assume without loss of generality that $\mathcal{A}$ is the dummy adversary, functioning as a communication intermediary between parties and the environment.

The simulator. We describe the UC simulator SIM for CHIC that will act as the ideal-world adversary, having access to the ideal functionality $\mathcal{F}_{\mathsf{PAKE}}$. SIM must simulate to $\mathcal{Z}$ protocol messages between honest participants without knowing the passwords chosen by $\mathcal{Z}$, while consistently answering random oracle and ideal cipher queries. In a limited number of cases, the simulator is unable to conclude the simulation and aborts. We argue in the proof that those bad events only happen with negligible probability and account for these events in the overall probability of $\mathcal{Z}$ distinguishing between the "real world" from the "ideal world".

- *First message*: After receiving $(\text{NewSession}, sid, P_i, P_j, Alice)$ from $\mathcal{F}_{\mathsf{PAKE}}$, SIM picks a random apk and sends message apk from P_i to P_j.
- *Second message*: After receiving $(\text{NewSession}, sid, P_j, P_i, Bob)$ from $\mathcal{F}_{\mathsf{PAKE}}$, SIM waits for a message apk sent to P_j from $\mathcal{A}$. Then SIM sets $fullsid \leftarrow (sid, P_i, P_j)$. In case the received apk is an output of the $\mathsf{m2F}$ that commits the adversary to a password pw, SIM extracts the password pw and tests it by sending $(\text{TestPwd}, sid, P_j, pw)$ to $\mathcal{F}_{\mathsf{PAKE}}$. If $\mathcal{F}_{\mathsf{PAKE}}$ replies with "correct guess", SIM computes the key according to the protocol specification and sends $(\text{NewKey}, sid, P_j, key)$ to $\mathcal{F}_{\mathsf{PAKE}}$; in all other cases (including "wrong guess",

```
A on (NewSession, $sid$, A, B, $pw$, $init$)
    $fullsid \leftarrow (sid, \mathsf{A}, \mathsf{B})$
    $(sk, pk) \leftarrow\!\!\$\ \mathsf{KEM.Keygen}(1^\lambda)$
    $(r, M) \leftarrow \mathsf{KEM.Split}(pk)$
    m2F:
        $R \leftarrow \mathsf{H}(fullsid, pw, r)$
        $T \leftarrow M \odot R$
        $t \leftarrow \mathsf{H}'(fullsid, pw, T)$
        $s \leftarrow \mathsf{IC.Enc}(t, r)$
    $apk \leftarrow (s, T)$

B on (NewSession, $sid$, B, A, $pw$, $resp$)
    $fullsid \leftarrow (sid, \mathsf{A}, \mathsf{B})$

                    Send ($apk$) →

B:
    $(s, T) \leftarrow apk$
    m2F$^{-1}$:
        $t \leftarrow \mathsf{H}'(fullsid, pw, T)$
        $r \leftarrow \mathsf{IC.Dec}(t, s)$
        $R \leftarrow \mathsf{H}(fullsid, pw, r)$
        $M \leftarrow T \odot R^{-1}$
    $pk \leftarrow \mathsf{KEM.Split}^{-1}(r, M)$
    $(c, K) \leftarrow\!\!\$\ \mathsf{KEM.Encap}(pk)$
    $tag \leftarrow \mathsf{H}_1(fullsid, pk, apk, c, K)$

                    ← Send ($c, tag$)

A:
    $K \leftarrow \mathsf{KEM.Decap}(sk, c)$
    if $tag \neq \mathsf{H}_1(fullsid, pk, apk, c, K)$
        $key \leftarrow\!\!\$\ \{0,1\}^\lambda$
    else
        $key \leftarrow \mathsf{H}_2(fullsid, pk, apk, c, K)$
    return $key$

B:
    $key \leftarrow \mathsf{H}_2(fullsid, pk, apk, c, K)$
    return $key$
```

Fig. 4. The CHIC protocol. KEM scheme has splittable public keys (Def. 4) with an efficiently computable and invertible map $\mathsf{Split} : \mathcal{PK}_\lambda \rightarrow N_\lambda \times G_\lambda$. The protocol makes use of a block cipher denoted as IC and hash functions H and H′ in an m2F configuration (Def. 5), with domains that align with Split and that are characterized by security parameter λ, i.e. $\{\mathsf{IC.Enc}, \mathsf{IC.Dec}\} : K_\lambda \times N_\lambda \rightarrow N_\lambda$, $\mathsf{H} : \{0,1\}^* \rightarrow G_\lambda$, $\mathsf{H}' : \{0,1\}^* \rightarrow K_\lambda$. Group operations within G are represented by $\odot$, and the inverse operation by $(\cdot)^{-1}$.

honest execution, etc.), SIM runs a $\mathsf{KEM.Keygen}$ algorithm, obtains a fresh key pair (pk, sk), computes the ciphertext c and the tag using the fresh pk, and sends $(\text{NewKey}, sid, P_j, \perp)$ to $\mathcal{F}_{\mathsf{PAKE}}$. To conclude the second message flow, SIM sends the message (c, tag) from P_j to P_i via $\mathcal{A}$.

- *Final output*: After receiving message (c, tag) sent to P_i from $\mathcal{A}$, in case of honest execution, SIM simply sends $(\text{NewKey}, sid, P_i, \perp)$ to $\mathcal{F}_{\mathsf{PAKE}}$. If message (c, tag) was tampered with by the adversary, SIM checks for a corresponding random oracle query to H_1 that returned tag. If such query has not been asked, SIM sends $(\text{TestPwd}, sid, P_i, \perp)$ and $(\text{NewKey}, sid, P_i, \perp)$ to $\mathcal{F}_{\mathsf{PAKE}}$, forcing a random session key. If tag comes from H_1, pk and apk are extracted. If appropriate queries were made to the $\mathsf{m2F}$, the password is also extractable. SIM extracts $\mathcal{A}$'s password guess pw and sends $(\text{TestPwd}, sid, P_i, pw)$ to $\mathcal{F}_{\mathsf{PAKE}}$. In case of a "correct guess", SIM computes the key by following the protocol and sends $(\text{NewKey}, sid, P_i, key)$ to $\mathcal{F}_{\mathsf{PAKE}}$. If tag is not valid (even if the password guess was correct) or $\mathcal{F}_{\mathsf{PAKE}}$ returned "wrong guess", SIM sends $(\text{NewKey}, sid, P_i, \perp)$ to $\mathcal{F}_{\mathsf{PAKE}}$. If the adversary did not commit to a password in its interaction with $\mathsf{m2F}$, SIM sends $(\text{TestPwd}, sid, P_i, \perp)$ and $(\text{NewKey}, sid, P_i, \perp)$ to $\mathcal{F}_{\mathsf{PAKE}}$.

Proof. We prove Theorem 1 via a series of game hops. The first game corresponds to a simulator that is not constrained in any way and executes the real world for the environment perfectly. Concretely, this simulator controls all inputs/outputs to the parties, as well as their communications with the environment. In each hop, we modify this simulator gradually, so that in the final game one can clearly see that it can be divided into two parts, where the first part corresponds to the ideal functionality $\mathcal{F}_{\mathsf{PAKE}}$ and the second part to the simulator described earlier, which has only black-box access to $\mathcal{F}_{\mathsf{PAKE}}$ and does not know the honest parties secret passwords. Conceptually, we think of $\mathcal{F}_{\mathsf{PAKE}}$ as always existing alongside our simulator and receiving the inputs from $\mathcal{Z}$: in the first game it is not used at all by the simulator, and gradually it will start using $\mathcal{F}_{\mathsf{PAKE}}$ to define the outputs of parties. Because the first game is identical to the real world and the last game is identical to the ideal world, we just need to show that the view of the environment is not affected by each of our modifications. Hence, in each hop, we analyze the probability of $\mathcal{Z}$ outputting 1 in the game G_i compared to that of $\mathcal{Z}$ outputting 1 in the game G_{i-1} and show that these change by a negligible amount.

Our analysis depends on the number of interactions between the environment and the execution model. To account for this, we consider and tally all queries made to the ideal cipher and random oracles, irrespective of whether they originate from honest parties or the adversary. We denote q_{IC} as the upper bound on queries to the ideal cipher, regardless of whether it is used for encryption ($\mathsf{IC.Enc}$) or decryption ($\mathsf{IC.Dec}$). Similarly, q_{H}, $q_{\mathsf{H}'}$, and q_{H_1} represent upper bounds on the number of queries made to the H, H', and H_1 oracles, respectively. Furthermore, we take into account the number of PAKE sessions and interactions occurring within each session. In this context, $q_{\mathsf{newSession}}$ serves as an upper bound on the number of sessions initiated by $\mathcal{Z}$, while q_{send} represents an upper bound on the number of messages delivered by $\mathcal{A}$ when interacting with the involved parties.

Game G_0 (Real world): Simulation perfectly mimics the world with oracles H, H', H_1, H_2, $\mathsf{IC.Enc}$ and $\mathsf{IC.Dec}$.

$$\Pr[\mathsf{G}_0] = \mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{CHIC}} \tag{5}$$

Game G_1 (Abort on random oracle collisions): On output collisions of H_1, H or H', the simulation aborts. This is a statistical hop with a birthday bound.

$$|\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_1]| \leq \frac{q_{\mathsf{H}_1}^2}{|\mathsf{Space}_{\mathsf{H}_1}|} + \frac{q_{\mathsf{H}}^2}{|\mathsf{Space}_{\mathsf{H}}|} + \frac{q_{\mathsf{H}'}^2}{|\mathsf{Space}_{\mathsf{H}'}|} \tag{6}$$

Game G_2 (Full domain sampling of IC and abort on collisions): On new $\mathsf{IC.Enc}$ and $\mathsf{IC.Dec}$ queries, simulator samples s and r regardless of previous answers and instead aborts on output collisions (even collisions across different keys). s and r are high-entropy, therefore this is a statistical hop with a negligible difference. Note that queries must be answered consistently and thus decrypting a ciphertext returned by $\mathsf{IC.Enc}$ or encrypting a plaintext returned by $\mathsf{IC.Dec}$, under the same key, is not considered a new query.

$$|\Pr[\mathsf{G}_1] - \Pr[\mathsf{G}_2]| \leq \frac{q_{\mathsf{IC}}^2}{|\mathsf{Space}_{\mathsf{IC}}|} \tag{7}$$

Game G_3 (Abort if a new sample for H' collides with a previous record of the IC): Upon sampling a new t (key for ideal cipher) for the simulation of H' oracle, if t is not fresh (and therefore already included in $\mathsf{List}_{\mathsf{IC}}$), the simulation aborts. This is a statistical hop.

$$|\Pr[\mathsf{G}_2] - \Pr[\mathsf{G}_3]| \leq \frac{q_{\mathsf{H}'} \cdot q_{\mathsf{IC}}}{|\mathsf{Space}_{\mathsf{H}'}|} \tag{8}$$

Game G_4 (Abort if a new sample for $\mathsf{IC.Dec}$ collides with a previous record of H): Upon sampling a new r for the simulation of $\mathsf{IC.Dec}$, if r is not fresh (and therefore already included in $\mathsf{List}_{\mathsf{H}}$), the simulation aborts. This is a statistical hop.

$$|\Pr[\mathsf{G}_3] - \Pr[\mathsf{G}_4]| \leq \frac{q_{\mathsf{IC}} \cdot q_{\mathsf{H}}}{|\mathsf{Space}_{\mathsf{IC}}|} \tag{9}$$

Game G_5 (On calls to $\mathsf{IC.Dec}$ where the password is extractable from the ideal cipher key, force a record to H): On a new query $\mathsf{IC.Dec}(t, s)$—i.e., a query where a fresh ideal cipher preimage r is sampled—check if t came out of H' oracle and, if so, introduce the following change to the oracle. First, extract the password pw associated with t (there is at most 1 since we have already discarded the possibility of collisions in H'), then call $\mathsf{H}(pw, r)$, forcing r to be added into the records of oracle H. Note that due to the abort triggers introduced in the previous games, this modification is equivalent to sampling a random pair (r, R)

and trying to program H directly by adding the tuple (pw, r, R) to List_H. This action will abort if either $(*, r, *) \in \mathsf{List}_\mathsf{H}$ (see Game G_4) or if $(*, *, R) \in \mathsf{List}_\mathsf{H}$ (see Game G_1). Nothing really changes unless $\mathsf{IC.Dec}$ triggers an abort that did not occur in the previous game. This is a statistical hop.

$$|\Pr[\mathsf{G}_4] - \Pr[\mathsf{G}_5]| \leq \frac{q_{\mathsf{IC}} \cdot q_{\mathsf{H}}}{|\mathsf{Space}_{\mathsf{IC}}|} + \frac{q_{\mathsf{IC}} \cdot q_{\mathsf{H}}}{|\mathsf{Space}_{\mathsf{H}}|} \tag{10}$$

Game G_6 (On calls to $\mathsf{IC.Dec}$ where the password is extractable from the ideal cipher key, use $\mathsf{KEM.Keygen}$ and store secrets): On a new query $\mathsf{IC.Dec}(t, s)$, if t came out of H' oracle, instead of directly sampling a random pair (r, R), the simulator relies on $\mathsf{KEM.Keygen}$ and $\mathsf{KEM.Split}$, and stores the secrets in $\mathsf{List}_\mathsf{secrets}$ for future use. If the adversary attempts to decrypt Alice's apk using her password, the record is also added to $\mathsf{List}_\mathsf{secrets}$. More precisely, if the adversary queries $\mathsf{IC.Dec}(t, s)$, where $t = \mathsf{H}'(fullsid, pw, T)$ and $apk = (s, T)$ is the message Alice sent in the session matching $fullsid$, and pw is Alice's password for that session, then add (pw, sk, pk, apk) to $\mathsf{List}_\mathsf{secrets}$, where (sk, pk) is Alice's key pair. This hop is down to the uniformity of KEM public keys.

$$|\Pr[\mathsf{G}_5] - \Pr[\mathsf{G}_6]| \leq q_{\mathsf{IC}} \cdot \mathsf{Adv}^{\mathsf{pk\text{-}uniformity}}_{\mathsf{KEM,Split}} \tag{11}$$

Game G_7 (Set random key via $\mathcal{F}_\mathsf{PAKE}$ if tag was not output by H_1): Modify Alice's response when tag was not output by H_1 wrt $fullsid$, apk and c: use $\mathcal{F}_\mathsf{PAKE}$ to generate the session key totally at random by compromising the session with an invalid password and then completing the session with NewKey. The protocol specification determines Alice's session key to be random if tag is incorrect.[15] A tag not coming out of H_1 will only be valid with negligible probability. Therefore, this is a statistical hop.

$$|\Pr[\mathsf{G}_6] - \Pr[\mathsf{G}_7]| \leq \frac{q_{\mathsf{send}}}{|\mathsf{Space}_{\mathsf{H}_1}|} \tag{12}$$

Game G_8 (For passive attacks, use a private oracle H_1^* without inputs pk and K to compute tag, and set session key directly via $\mathcal{F}_\mathsf{PAKE}$ instead of using the key coming from H_2): For passive attacks, i.e. messages are correctly computed and forwarded to the intended party (apk from Alice to Bob, and possibly (c, tag) from Bob to Alice), compute the tag with private oracle H_1^* and use the functionality to generate the session key, without testing the password.

The intuition of this hop is that the KEM ciphertext must conceal K, therefore the adversary will not call $\mathsf{H}_1(*, *, *, *, K)$ nor $\mathsf{H}_2(*, *, *, *, K)$. If it does, the simulator breaks the one-wayness of the KEM. The technical difficulty in the reduction is that the simulator does not know ahead of time if the session will be actively attacked. Therefore, it must embed the challenge pk^* in each session, one at a time (hybrid argument), and complete the simulation without detectable changes to the protocol. If the adversary relays correctly apk from Alice to Bob

[15] Our protocol is implicitly rejecting to follow the standard $\mathcal{F}_\mathsf{PAKE}$ functionality.

but then decides to actively interfere with the communication and forward its own (c, tag) back to Alice, the simulator faces the dilemma of whether to force Alice to use a random session key (if tag is invalid) or the session key resulting from $\mathsf{H}_2(fullsid, pk, apk, c, K)$ (if tag is valid). This boils down to whether c encrypts K included in tag or not. However, because we embedded the challenge pk^* to compute the first flow of messages, we no longer can decrypt c. For this reason, we reduce this hop down to OW-PCA and take advantage of the PCO oracle to check if the key K included in the tag is effectively the key encrypted under c.

The reduction goes as follows (hybrid argument, one public key at a time): i. Embed challenge pk^* into Alice's initialization procedure. ii. If the adversary is passive and delivers apk to Bob, reduction uses challenge c^* and private oracles H_1^* and H_2^* to proceed. These private oracles receive the same inputs as their public counterparts H_1 and H_2, except for the arguments pk and K. (Note that K^* encrypted under c^* is unknown to the reduction.) Since the ciphertext c^* is an input to both H_1 and H_2, this fixes a single key anyway and the games are identical unless K^* is queried to either oracle. If such a query is never placed, the usage of these private oracles is independent of $\mathcal{Z}$'s view. iii. On Alice's side, if the adversary is still passive, decryption is not needed: tag is valid and session key is derived from private oracle H_2^*. Due to the uniqueness of inputs, private oracle H_2^* will produce the same key on both sides, as would the public oracle H_2 in G_7 and NewKey query to $\mathcal{F}_{\mathsf{PAKE}}$ in G_8.

If the adversary is active (and the reduction embedded the challenge pk^* in this session) the reduction algorithm will use the PCO oracle to verify the tag: it verifies that the unique H_1 entry corresponding to the tag includes key encrypted under c^*. In this reduction, there is at most one PCO call per embedded challenge public key pk^* since KEM decryption occurs only in one place in our protocol. If this check fails, the reduction returns a fresh random key to the attacker, which is consistent with both games: trivially so in G_7, and in G_8 because this forces the functionality to produce a fresh random session key by issuing a TestPwd with $\perp$.

When the adversary concludes its run, the reduction algorithm confirms the inclusion of the correct K^* in queries to H_1 and H_2 via calls to the PCO oracle before submitting its answer against the one-wayness property of KEM ciphertexts. If no such query with the correct K^* exists, G_7 and G_8 are identical. As an alternative approach, randomly selecting an entry from the H_1 and H_2 tables can lead to a less tight reduction. However, this approach does not require confirmation of the inclusion of K^* in H_1 and H_2 oracle queries, and therefore at most one query to the PCO oracle is required for the correct simulation of active attacks to Alice, after embedding the challenge public key in the first flow from Alice to Bob. Importantly, Lemma 1 establishes the equivalence between OW-1PCA and OW-CPA.

$$|\Pr[\mathsf{G}_7] - \Pr[\mathsf{G}_8]| \leq q_{\mathsf{send}} \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow\text{-}pca}} \tag{13}$$

$$|\Pr[\mathsf{G}_7] - \Pr[\mathsf{G}_8]| \leq (q_{\mathsf{H}_1} + q_{\mathsf{H}_2}) \cdot q_{\mathsf{send}} \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow\text{-}cpa}} \tag{14}$$

Game G_9 (Simulate Bob's response with a fresh public key for passive attacks): For honestly transmitted *apk*, the simulator creates ciphertext c with a fresh public key, then computes *tag* with the private oracle H_1^* (as in the previous game), and finally sends (c, tag) on Bob's behalf. We bridge this hop using ANO-1PCA, with a hybrid argument, replacing one public key at a time. Note that Alice does not decrypt honestly transmitted ciphertexts since G_8. However, if there's an active attack on the second round of the session where the reduction programmed the challenge pk_0 from ANO-1PCA game, the decryption key is not available. As before, the reduction takes advantage of a single call to the PCO oracle and the (single) relevant record of H_1 to determine whether the *tag* is valid.

More in detail, the reduction goes as follows (hybrid argument, one public key at the time): i. Embed challenge pk_0 into Alice's initialization procedure. ii. If the adversary is passive and delivers *apk* to Bob, reduction uses challenge c^*. iii. On Alice's side, if the attack is passive, no need to decrypt c^*. If there is an active attack, extract K from *tag* by inspecting H_1 records, and check K against c submitted by the adversary and sk_0 to determine the validity of *tag* without actually decrypting the ciphertext. Note that the PCO oracle of the standard ANO-1PCA game allows checks against both sk_0 and sk_1, and the reduction embedded pk_0 on Alice's side. Therefore, checks must be carried out against sk_0.

There are a few noteworthy observations regarding the definitional requirements for this reduction. For starters, we only need a weaker version of ANO-1PCA where the PCO oracle only allows plaintext checks against one of the secret keys. Another observation is that we don't require the challenger of the ANO-1PCA game to provide K^* as part of the challenge. This is a direct result of the modification introduced in G_8 that lifts the need to use the encapsulated key for passive attacks (via the usage of private oracle H_1^* to compute the tag, and NewKey to $\mathcal{F}_{\mathsf{PAKE}}$ to set the session key).[16] This reduction algorithm perfectly interpolates between games G_8 and G_9. If challenge c^* is a result of KEM.Encap with pk_0, this corresponds to G_8. On the other hand, if c^* is a result of KEM.Encap with pk_1, the simulation adheres to the specifications of G_9.

$$|\Pr[\mathsf{G}_8] - \Pr[\mathsf{G}_9]| \leq q_{\mathsf{send}} \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ano\text{-}1pca}} \tag{15}$$

Game G_{10} (Active attacks on Alice: The tag is invalid if the password cannot be extracted from an adversarially crafted message from Bob to Alice): On adversarially crafted (c, tag) sent to Alice, the *tag* forces a commitment to a single *pk* and, consequently, to a unique password due to the joint operation of IC.Dec and H'. The only case in which password extraction fails is if the adversary did not reconstruct the pk to which it committed using calls to IC.Dec and H'. However, in this case, the correct pk that Alice will be using is information-theoretically hidden from the adversary's view. More in detail, in G_{10} we check if the *pk* was not obtained from *apk* via the appropriate calls to

[16] We note that this is the point in the proof where we could not find a way to avoid a decryption-like oracle and that forces us to use an actively secure KEM.

H' and then $\mathsf{IC.Dec}$. (Note that since G_6 the appropriate decryption calls create a record in $\mathsf{List}_{\mathsf{secrets}}$.) If this is not the case, then *tag* is declared as invalid. In such cases, we force a random session key via $\mathcal{F}_{\mathsf{PAKE}}$.

The two games G_9 and G_{10} are identical unless the adversary guessed Alice's public key (and created the *tag* sent to Alice with it) without having obtained it from *apk* via the appropriate calls to H' and then $\mathsf{IC.Dec}$. The probability of guessing Alice's public key by chance is tied to the min-entropy k of the public keys generated by $\mathsf{KEM.Keygen}$. This is not a new assumption on the properties of KEM as it follows from $\mathsf{UNI\text{-}PK}$ that each part of the split public key has at least min-entropy λ. We denote ϵ as a negligible function.

$$|\Pr[\mathsf{G}_9] - \Pr[\mathsf{G}_{10}]| \leq \epsilon(\lambda) \qquad (16)$$

Game G_{11} (Active attacks on Alice: If the password is extractable, test it and proceed accordingly): On Alice's side, if the password can be extracted, test the password via TestPwd. If the guess is correct, run the protocol honestly and program the session key. If the guess is wrong, tell functionality to complete the session with a random key. Note that different passwords are guaranteed to lead to different public keys for a fixed *apk*, as oracles discard collisions. In turn, because *pk* is also included as an argument of H_1, the tag-verification procedure is bound to fail. Therefore, Game G_{10} and Game G_{11} are identical from $\mathcal{Z}$'s perspective. This is a bridge hop.

$$\Pr[\mathsf{G}_{10}] = \Pr[\mathsf{G}_{11}] \qquad (17)$$

Game G_{12} (Simulate Alice's initial message without using the password): Notice that the simulator deals with Alice's response without using *sk*, except for the case where Alice is actively attacked with the correct password. Therefore, the simulator can simulate a NewSession for Alice by directly sampling *apk*, leaving the generation of the public key for later. As such, the password *pw* is not required at this stage.

Nevertheless, the simulator of G_{12} creates an IC record for $apk := (s, T)$, with placeholders that can later be replaced by Alice's *pk*. More precisely, it adds $(\perp, \perp, s, mode = E)$ to $\mathsf{List}_{\mathsf{IC}}$. If s is unfresh, the simulation aborts. The record only gets updated when Alice's password *pw* is confirmed to be correct as a result of a TestPwd query to $\mathcal{F}_{\mathsf{PAKE}}$, and $\mathsf{m2F}_{pw}^{-1}(apk)$ is computed by querying its oracles. Recall that queries to $\mathsf{IC.Dec}$ with t that permits password extraction leads to *sk* being embedded in $\mathsf{List}_{\mathsf{secrets}}$. So, the decryption key is always available in the only case still needed (active attack with the correct password).

Following the rules of the previous game G_{11}, Alice generates a key-pair with $\mathsf{KEM.Keygen}$ and then computes *apk* by feeding *pk* to the oracles of the $\mathsf{m2F}$, which leads to early abortion if the newly sampled R is in $\mathsf{List}_{\mathsf{H}}$ (rule added in G_1), if the newly sampled t is in $\mathsf{List}_{\mathsf{IC}}$ (rule added in G_3), and if the newly sampled s is in $\mathsf{List}_{\mathsf{IC}}$ (rule added in G_2). In G_{12}, we abort only if the newly sampled s is in $\mathsf{List}_{\mathsf{IC}}$. This means that the other abortion events have to be accounted for in the analysis of this game hop. Furthermore, if the adversary places a new query to $\mathsf{IC.Enc}$ and happens to land on s —it's important to emphasize *new query*, meaning this only applies to queries $\mathsf{IC.Enc}(t, r)$ where r is not the result of a previous query $\mathsf{IC.Dec}(t, s)$,— the game aborts since there's already a record (albeit incomplete). Notice that in G_{11} there is one particular value of r for which the oracle will respond without aborting–this is the r obtained by splitting Alice's *pk*. The probability of guessing r by chance is tied to the min-entropy k of the first element resulting from the split of public keys generated by KEM.Keygen. Again, this is not a new assumption about KEM, as it follows from UNI-PK.

$$|\Pr[\mathsf{G}_{11}] - \Pr[\mathsf{G}_{12}]| \leq \frac{q_{\mathsf{newSession}} \cdot q_{\mathsf{H}}}{|\mathsf{Space}_{\mathsf{H}}|} + \frac{q_{\mathsf{newSession}} \cdot q_{\mathsf{H}'}}{|\mathsf{Space}_{\mathsf{H}'}|} + \epsilon(\lambda) \tag{18}$$

Game G_{13} (Active attacks on Bob: if there's no record consistent with *apk* having been computed in the forward direction, use private oracle H_1^* to compute *tag* an set random session key via $\mathcal{F}_{\mathsf{PAKE}}$): The attacker sends its own *apk* to Bob and there is no record consistent with *apk* having been computed in the forward direction. The term *forward direction* refers to the encryption direction within the ideal cipher, which is why the simulator tracks how the records of the IC were generated. Formally, $apk = (s, T)$ was computed in the forward direction if there exist a record $(t, *, s, mode = E) \in \mathsf{List}_{\mathsf{IC}}$ such that there is a record $(fullsid, *, T, t) \in \mathsf{List}_{\mathsf{H}'}$ wrt the unique $fullsid$ associated with the Alice instance receiving *apk*. In such cases, the simulator uses the private oracle H_1^* to compute the *tag* and sets a random session key via $\mathcal{F}_{\mathsf{PAKE}}$. Recall that the private oracle H_1^* does not take as input *pk* and K. We reduce this hop down to OW-PCA. We use a hybrid argument, changing the behavior of one Bob session at a time. The intuition is that if *apk* was not computed in the forward direction with an appropriate call to IC.Enc, the attacker has no control over the KEM public key (and corresponding secret key) associated with *apk* sent to Bob. Therefore, the attacker cannot decrypt Bob's ciphertext, and is unlikely to query H_1 with K encrypted in Bob's response. If it does, we break the OW-PCA game of KEM.

The reduction algorithm knows Bob's password. The inverse of the attacker's *apk* sent to Bob, under Bob's password, must be the challenge pk^* of the OW-PCA game. The difficulty in arguing this hop arises from the adversary's potential actions with *apk*: they might attempt to decrypt it using Bob's password before or after sending it, or they may not decrypt *apk* with Bob's password at all (willingly or because the Bob's password was never correctly guessed by the adversary).

Remember, in this particular game hop, we are exclusively handling adversary-generated *apk* values, which are not computed following the forward direction of the m2F. Therefore, we apply a hybrid argument over all q_{send} queries from Alice to Bob, and all IC.Dec queries, carefully associating the challenge pk^* with one of these queries. The reduction algorithm loses the ability to decrypt ciphertexts encrypted under pk^*, but in the protocol only Alice needs to decrypt ciphertexts and she will do so under her secret key (regardless of whether *apk* sent out is crafted by the adversary and possibly associated with pk^*).

The reduction algorithm also embeds c^* in the computation of *tag* with private oracle H_1^* and in Bob's response. It also monitors queries to public oracles H_1 and H_2, extracting K and testing with the PCO oracle against challenge c^*. If the PCO oracle returns *true*, the reduction would submit K and would win the OW-PCA game. Otherwise, the usage of private oracle H_1^* and setting Bob's session key to be random via $\mathcal{F}_{\mathsf{PAKE}}$ is identical from $\mathcal{Z}$'s view.

Alternatively, as also described in the proof strategy of the hop to G_8, if we are willing the bear the cost of a loss in tightness, we could use a guessing argument instead by simply outputting a K queried to one of the public oracles H_1 and H_2, and avoid relying on any PCO oracle for this reduction (as mentioned earlier, Alice is always able to decrypt).

$$|\Pr[\mathsf{G}_{12}] - \Pr[\mathsf{G}_{13}]| \leq (q_{\mathsf{send}} + q_{\mathsf{IC}}) \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow\text{-}pca}} \tag{19}$$

$$|\Pr[\mathsf{G}_{12}] - \Pr[\mathsf{G}_{13}]| \leq (q_{\mathsf{H}_1} + q_{\mathsf{H}_2}) \cdot (q_{\mathsf{send}} + q_{\mathsf{IC}}) \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow\text{-}cpa}} \tag{20}$$

Game G_{14} (Active attacks on Bob: if there's no record consistent with *apk* having been computed in the forward direction, encrypt the ciphertext under a freshly generated public key): As in the case of the previous game hop, the attacker sends its own *apk* to Bob and there's no record consistent with *apk* having been computed in the forward direction. Now, the simulator encrypts the ciphertext that Bob sends out under a freshly generated public key. This is a reduction to ANO-CPA.

The reduction is similar to the previous game hop in that we embed pk_0 in one send query to Bob at the time, and then embed the challenge c^* in Bob's response. As in the analysis of the previous game hop, we have to account for the possibility that the attacker tried to decrypt *apk* under Bob's password before sending it. In that case, pk_0 needs to be embedded upon the IC.Dec call. In the worst case, the lost in tightness w.r.t. to ANO-CPA is limited by $q_{\mathsf{send}} + q_{\mathsf{IC}}$. If c^* was encrypted under pk_0, we adhere to the specifications of G_{13}. If it was encrypted under pk_1, we adhere to the rules of G_{14}. We have already established in the previous game that *tag* is computed via private oracle H_1 (that does not take *pk* as input). As in the reduction strategy of the previous game hop, the challenge pk^* of ANO-CPA is never associated with the *apk* sent by Alice. Thus, Alice's decryption key *sk* is always available when needed, and a PCO oracle is also not needed for this reduction.

$$|\Pr[\mathsf{G}_{13}] - \Pr[\mathsf{G}_{14}]| \leq (q_{\mathsf{send}} + q_{\mathsf{IC}}) \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ano\text{-}cpa}} \tag{21}$$

Game G_{15} (Active attacks on Bob: if there is a record consistent with *apk* having been computed in the forward direction, extract the password, test it, and use private oracle H_1^* and set a random session key if "wrong guess"): The simulator now deals with the case where there is a record consistent with *apk* having been computed in the forward direction. The simulator extracts the password and tests it. If "correct guess", the simulator keeps following the protocol and sets the correctly-computed session key via $\mathcal{F}_{\mathsf{PAKE}}$ (this doesn't change anything from $\mathcal{Z}$'s view). If "wrong guess", the simulator makes use of private oracle H_1^* to compute the *tag* and sets a random session key via $\mathcal{F}_{\mathsf{PAKE}}$. The reduction is similar to that of G_{13}.

$$|\Pr[\mathsf{G}_{14}] - \Pr[\mathsf{G}_{15}]| \leq (q_{\mathsf{send}} + q_{\mathsf{IC}}) \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow\text{-}pca}} \tag{22}$$

$$|\Pr[\mathsf{G}_{14}] - \Pr[\mathsf{G}_{15}]| \leq (q_{\mathsf{H}_1} + q_{\mathsf{H}_2}) \cdot (q_{\mathsf{send}} + q_{\mathsf{IC}}) \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow\text{-}cpa}} \tag{23}$$

Game G_{16} (Active attacks on Bob: if there is a record consistent with *apk* having been computed in the forward direction, extract the password, test it, and encrypt the ciphertext under a freshly generated public key if "wrong guess"): This change and reduction is similar to that argued in G_{14}.

$$|\Pr[\mathsf{G}_{15}] - \Pr[\mathsf{G}_{16}]| \leq (q_{\mathsf{send}} + q_{\mathsf{IC}}) \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ano\text{-}cpa}} \tag{24}$$

Game G_{17} (Ideal world): At this point, we are in the ideal world, where the simulator is using the ideal functionality $\mathcal{F}_{\mathsf{PAKE}}$ to generate all keys except for those where there is a correct password guess.

$$\Pr[\mathsf{G}_{16}] = \Pr[\mathsf{G}_{17}] = \mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathcal{F}_{\mathsf{PAKE}}} \tag{25}$$

Bringing all these elements together and assuming KEM is a $\mathsf{OW\text{-}PCA}$-secure key encapsulation mechanism, we obtain the result shown in Equation 26. For the sake of completeness, a description in pseudo-code of the simulator SIM of the ideal world is provided in Appendix A. Each step of the process, starting from the code execution of uncorrupted parties in the real world and leading to the simulation of the ideal world, is meticulously detailed. Every modification is framed and cross-referenced with the specific game hop where it was initially introduced to ensure a traceable progression of the proof.

$$\begin{aligned}
&|\mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{CHIC}} - \mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathcal{F}_{\mathsf{PAKE}}}| \leq \\
&\qquad q_{\mathsf{IC}} \cdot \mathsf{Adv}_{\mathsf{KEM},\mathsf{Split}}^{\mathsf{pk\text{-}uniformity}} \\
&+ \quad (3 \cdot q_{\mathsf{send}} + 2 \cdot q_{\mathsf{IC}}) \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ow\text{-}pca}} \\
&+ \quad (3 \cdot q_{\mathsf{send}} + 2 \cdot q_{\mathsf{IC}}) \cdot \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{ano\text{-}1pca}} \\
&+ \quad \frac{q_{\mathsf{IC}}^2 + 2 \cdot q_{\mathsf{IC}} \cdot q_{\mathsf{H}}}{|\mathsf{Space}_{\mathsf{IC}}|} + \frac{q_{\mathsf{H}_1}^2 + q_{\mathsf{send}}}{|\mathsf{Space}_{\mathsf{H}_1}|} \\
&+ \quad \frac{q_{\mathsf{H}}^2 + q_{\mathsf{IC}} \cdot q_{\mathsf{H}} + q_{\mathsf{newSession}} \cdot q_{\mathsf{H}}}{|\mathsf{Space}_{\mathsf{H}}|} \\
&+ \quad \frac{q_{\mathsf{H}'}^2 + q_{\mathsf{IC}} \cdot q_{\mathsf{H}'} + q_{\mathsf{newSession}} \cdot q_{\mathsf{H}'}}{|\mathsf{Space}_{\mathsf{H}'}|} + \epsilon(\lambda)
\end{aligned} \tag{26}$$

□

On tightness. The bounds we give here are aligned with those obtained in prior works on EKE-like constructions from KEMs. The main difference with respect to Diffie-Hellman based constructions is that we cannot use self reducibiliy properties to remove the multiplicative factors associated with dealing with multiple-instance KEM security properties. Intuitively, the q_{IC} multiplicative factor is the most problematic, but it seems intrinsic to the use of the ideal cipher: it corresponds to the reduction's uncertainty as to which of the adversary's reverse ideal cipher queries will the adversary choose to fix the KEM public key on which it will be challenged. A KEM with a tight proof of multi-instance security would solve this problem.

Implications for the proofs in [27,4]. In game G_9 we explain why at that point in the proof we could not avoid making a decryption-like query, and that forces us to use an actively secure KEM. Furthermore, the 1 PCA query needed for the reduction seems applicable regardless of whether the protocol uses IC [4], HIC [27], or directly m2F [here], to password-encrypt the pk. This stands in contention with the results in [27,4], where we believe the authors have missed this point. A guessing strategy does not work for indistinguishability-based definitions for the reasons discussed in "CPA versus iPCA" subsection. We also note that ANO-PCA was already used in the security proof from Pan and Zeng [26]. The authors claim that OCAKE protocol in [4] lacks *perfect forward secrecy* (PFS), but it is unclear whether the claim is attributed to the fact that the original proof for the protocol requires the underlying KEM to satisfy merely ANO-CPA.

6 Implementation and Performance Analysis

We make two preliminary notes on our instantiation of CHIC, which distinguish this work from previous proposals for building PAKE from a lattice-based KEM in the Ideal Cipher model.

Firstly, contrary to what has been suggested in previous papers [4,27], our security proof shows that the construction requires a KEM that offers more than just passive security (namely ANO-1PCA). For this reason, we take the (CCA-secure) Kyber standard defined in FIPS 203 [24] as the natural off-the-shelf lattice-based KEM instantiation. Indeed, Kyber has been shown to be IND-CCA [8,28] and ANO-CCA secure in [15,20,29].

Secondly, we recall that the bandwidth requirements of the IND-CCA version of Kyber are the same as that of the underlying IND-CPA construction: this is one of the properties of the Fujisaki-Okamoto transformation used by Kyber. For this reason, when it comes to bandwidth usage, our construction still outperforms previous proposals that (unjustifiably) propose to use the IND-CPA version. Indeed, there is no overhead in public-key transmission in the first flow of our protocol due to the compact half-ideal cipher, whereas in the second flow we have only the overhead of transmitting the (short) MAC tag.

It remains to show that the Kyber KEM satisfies the remaining requirements of having splittable and pseudo-uniform public keys. We refer to Appendix C for a detailed proof that Kyber indeed satisfies the conditions outlined in Def. 4.

Our Implementation. We have implemented CHIC in C by extending the reference implementation of Kyber available from `github.com/pqcrystals/kyber`. The implementation is provided as supplementary material.

Before discussing parameter choices and giving some performance figures, we briefly describe how we implemented the three components of the compact half-ideal cipher construction, as well as the computation of the MAC tag and key derivation hashes, which are all that's needed beyond Kyber KEM:

- **Ideal cipher over 256-bits:** We take the Rijndael variant that uses 256-bit blocks and 256-bit keys[17]. The code was taken from the open-source tool *ccrypt*, which in turn adapts on the original Rijndael reference implementation. We recall that this block cipher is used to hide the seed component ρ that results from public-key splitting.
- **Hashing to the Kyber polynomial ring R_q:** We reuse the implementation of the rejection sampling procedure that is used internally by Kyber to expand the public-key seed to a $k \times k$ matrix over R_q. The only difference to the Kyber implementation is that, rather than sampling a $k \times k$ matrix starting from a seed ρ, our implementation samples a vector of size k, seeded by the input to random oracle H in our m2F construction. We recall that the output of this procedure is used to mask the vector over R_q^k that results from public-key splitting using a group operation.
- **Masking vectors in R_q^k:** We reuse the functions already available in the Kyber code that permit adding and subtracting vectors over R_q^k.
- **Hashing to the key space of** Rijndael: We use SHA3-256 to produce the required 32-bytes.
- **Key Derivation Function and Tag Computation** Since these two hash functions take the same input, we implement them as a single SHA3-512 computation that produces 64 bytes, which we then split to obtain the session key (which is kept secret) and the tag (which is transmitted).

Parameter selection. We consider all three variants of Kyber (Kyber512, Kyber768 and Kyber1024) as plausible instantiations for KEM. In our security analysis, we exclude the possibility of collisions on the block cipher, as well as on the hash function that derives the block cipher key. For this reason, we opt to use Rijndael with 256-bit block and key sizes. As a result, our parameter selection ensures at least 128-bit security against classical adversaries. We extend the discussion on the guarantees provided by CHIC in the conclusion section.

Performance Analysis. The bandwidth overhead of our protocol over Kyber KEM is minimal: it comprises of only 32-bytes for the tag in the second flow. Concerning execution time, Table 1 shows values in microseconds for the two

[17] Alternatively, we could employ standard AES and extend its domain from 128-bit to 256-bit blocks using the 3-round Feistel domain extender from [14], which is also indifferentiable from IC if AES itself is indifferentiable from IC.

stages of the initiator and the single stage of the responder for three cases: 1) using just the IND-CPA version of Kyber; 2) using just the IND-CCA version of Kyber; and 3) using CHIC. The measurements were taken in a modest laptop with a 2.3 GHz Intel "Core i5" processor with four cores, 128 MB of embedded eDRAM, a 6 MB shared level 3 cache, and 16GB of RAM. We did not explore aggressive optimizations using parallelism (or even SIMD implementations), so these results can definitely be improved. The overhead in computation time for initiators is around 25% for Kyber 768 wrt the bare CCA KEM key exchange. For responders, it is around 50%. Overall, these overheads decrease as the security level of Kyber increases, but the execution times are still in the order of tens of microseconds.

Table 1. Experimental results in microseconds. Comparison of execution times of CHIC participants (two initiator stages and responder single stage) with respect to key exchange using only a CPA or CCA Kyber KEM.

	CPA KEM			CCA KEM			CHIC		
	KeyGen	Enc	Dec	KeyGen	Enc	Dec	Start	Resp	End
Kyber512	25	29	9	45	49	12	70	74	14
Kyber768	28	36	41	49	59	65	75	85	93
Kyber1024	36	56	53	61	87	83	89	123	117

7 Conclusion

It's important to underscore the significance of the 30+ years of research enabling CHIC's development. The concept of constructing PAKE by encrypting a public key with a password dates back to EKE [6]. However, EKE and its variant OEKE [11] upon which EKE-KEM [27] and CHIC are based, were never standardized or deployed because of the difficulty of encrypting group elements with a password while preventing offline dictionary attacks. The work of [27] cleverly sidesteps the use of the IC over groups by using the m2F as a randomized cipher, which increases ciphertext size. CHIC avoids this ciphertext expansion by splitting Kyber pk and using the m2F instead as a keyed PRP. Our implementation represents the first practical realization of an approach previously deemed impractical [17], and it leverages the just-concluded ML-KEM standard [24].

This brings up the question: What security guarantees does CHIC provide against a quantum adversary? We proved that CHIC UC-realizes $\mathcal{F}_{\mathsf{PAKE}}$ in the IC and RO model. The UC framework is the gold standard security definition for PAKE because it captures arbitrary password distributions. Unfortunately, it's known that $\mathcal{F}_{\mathsf{PAKE}}$ cannot be UC-realized without an idealized computation model or CRS [12]. In our implementation we instantiate the IC with Rijndael-256 and the RO with SHA3. These building blocks are quantum-resistant, but our security proof does not allow the adversary to query the RO and IC in superposition. Sufficiently large quantum computers don't yet exist, but adversaries today can actively attack the protocol with classical computing capabilities and store PAKE transcripts now, hoping to decrypt them later with a quantum computer. However, such decryption would require breaking the KEM, and Kyber is quantum-resistant. This puts CHIC in the category of protocols secure against the "harvest-now/decrypt-later" quantum threat.

With a UC proof in the plain model aside, a consolidated analysis against adversaries that can actively attack the protocol with the support of a quantum computer, demands working in the Quantum Random Oracle Model (QROM) [7] and/or Quantum Ideal Cipher Model (QICM) [18], which is a future direction to pursue. The QROM is better understood than the QICM [18]. To the best of our knowledge, the only PAKE protocols analysed in the UC framework and the QROM were only very recently proposed in [19]. While the authors' contribution is a significant step forward, a modular approach based on standard KEMs is likely yield a better candidate as a practical substitute of currently widely-used PAKE protocols. In this respect, CHIC is extremely efficient, with minimal overhead compared to the KEM it's based on.

Acknowledgements

We thank the anonymous reviewers of ASIACRYPT'24 for their valuable feedback. Afonso Arriaga and Marjan Škrobot received support from the Luxembourg National Research Fund (FNR) under the CORE Junior project (C21/IS/16236053/FuturePass).

References

1. Abdalla, M., Bellare, M., Neven, G.: Robust encryption. In: Theory of Cryptography – TCC 2010. pp. 480–497. Springer (2010)
2. Abdalla, M., Haase, B., Hesse, J.: Security analysis of CPace. In: Advances in Cryptology – ASIACRYPT 2021. pp. 711–741. Springer (2021)
3. Alnahawi, N., Hövelmanns, K., Hülsing, A., Ritsch, S., Wiesmaier, A.: Towards post-quantum secure PAKE - A tight security proof for OCAKE in the BPR model. Cryptology ePrint Archive, Paper 2023/1368 (2023), https://eprint.iacr.org/2023/1368
4. Beguinet, H., Chevalier, C., Pointcheval, D., Ricosset, T., Rossi, M.: GeT a CAKE: Generic transformations from key encaspulation mechanisms to password authenticated key exchanges. In: Applied Cryptography and Network Security – ACNS 2023. pp. 516–538. Springer (2023)
5. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Advances in Cryptology – EUROCRYPT 2000. pp. 139–155. Springer (2000)
6. Bellovin, S., Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: Symposium on Research in Security and Privacy – S&P 1992. pp. 72–84. IEEE Computer Society (1992)
7. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Advances in Cryptology – ASIACRYPT 2011. pp. 41–69. Springer (2011)

8. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehle, D.: CRYSTALS - Kyber: A CCA-secure module-lattice-based KEM. In: European Symposium on Security and Privacy – EuroS&P 2018. pp. 353–367. IEEE Computer Society (2018)
9. Boyko, V., MacKenzie, P., Patel, S.: Provably secure password-authenticated key exchange using Diffie-Hellman. In: Advances in Cryptology – EUROCRYPT 2000. pp. 156–171. Springer (2000)
10. Bradley, T., Camenisch, J., Jarecki, S., Lehmann, A., Neven, G., Xu, J.: Password-authenticated public-key encryption. In: Applied Cryptography and Network Security – ACNS 2019. pp. 442–462. Springer (2019)
11. Bresson, E., Chevassut, O., Pointcheval, D.: Security proofs for an efficient password-based key exchange. In: ACM Conference on Computer and Communications Security – CCS 2003. pp. 241–250. Association for Computing Machinery (2003)
12. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. In: Advances in Cryptology – EUROCRYPT 2005. pp. 404–421. Springer (2005)
13. Coron, J.S., Dodis, Y., Mandal, A., Seurin, Y.: A domain extender for the ideal cipher. In: Theory of Cryptography – TCC 2010. pp. 273–289. Springer (2010)
14. Coron, J.S., Dodis, Y., Mandal, A., Seurin, Y.: A domain extender for the ideal cipher. In: Theory of Cryptography – TCC 2010. pp. 273–289. Springer (2010)
15. Grubbs, P., Maram, V., Paterson, K.G.: Anonymous, robust post-quantum public key encryption. In: Advances in Cryptology – EUROCRYPT 2022. pp. 402–432. Springer (2022)
16. Guo, C., Lin, D.: Improved domain extender for the ideal cipher. Cryptography and Communications **7**(4), 509–533 (2015)
17. Hao, F., van Oorschot, P.C.: SoK: Password-authenticated key exchange – theory, practice, standardization and real-world lessons. In: ACM Asia Conference on Computer and Communications Security – AsiaCCS 2022. pp. 697–711. Association for Computing Machinery (2022)
18. Hosoyamada, A., Yasuda, K.: Building quantum-one-way functions from block ciphers: Davies-meyer and merkle-damgård constructions. In: Advances in Cryptology – ASIACRYPT 2018. pp. 275–304. Springer (2018)
19. Lyu, Y., Liu, S., Han, S.: Universal composable password authenticated key exchange for the post-quantum world. In: Advances in Cryptology – EUROCRYPT 2024. pp. 120–150. Springer (2024)
20. Maram, V., Xagawa, K.: Post-quantum anonymity of Kyber. In: Public-Key Cryptography – PKC 2023. pp. 3–35. Springer (2023)
21. Maurer, U., Renner, R., Holenstein, C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Theory of Cryptography – TCC 2004. pp. 21–39. Springer (2004)
22. McQuoid, I., Rosulek, M., Roy, L.: Minimal symmetric PAKE and 1-out-of-n OT from programmable-once public functions. In: ACM Conference on Computer and Communications Security – CCS 2020. pp. 425–442. Association for Computing Machinery (2020)
23. Naehrig, M., Alkim, E., Bos, J., Ducas, L., Easterbrook, K., LaMacchia, B., Longa, P., Mironov, I., Nikolaenko, V., Peikert, C., Raghunathan, A., Stebila, D.: FrodoKEM. Tech. rep., National Institute of Standards and Technology (2020), available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions`

24. NIST: FIPS203, Module-Lattice-based Key-Encapsulation Mechanism Standard. Federal Information Processing Standards Publication (2023), `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf`
25. Okamoto, T., Pointcheval, D.: REACT: Rapid enhanced-security asymmetric cryptosystem transform. In: Topics in Cryptology – CT-RSA 2001. pp. 159–174. Springer (2001)
26. Pan, J., Zeng, R.: A generic construction of tightly secure password-based authenticated key exchange. In: Advances in Cryptology – ASIACRYPT 2023. pp. 143–175. Springer (2023)
27. Santos, B.F.D., Gu, Y., Jarecki, S.: Randomized half-ideal cipher on groups with applications to UC (a)PAKE. In: Advances in Cryptology – EUROCRYPT 2023. pp. 128–156. Springer (2023)
28. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehlé, D., Ding, J.: CRYSTALS-KYBER. Tech. rep., National Institute of Standards and Technology (2022), available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`
29. Xagawa, K.: Anonymity of NIST PQC round 3 KEMs. In: Advances in Cryptology – EUROCRYPT 2022. pp. 551–581. Springer (2022)

A The simulator

We detail the ideal world simulator SIM for proof of Theorem 1, using labeled frames to indicate the game hops where modifications occur.

```
On query $\mathsf{H}_1(fullsid, pk, apk, c, K)$:
    find $(\mathbf{fullsid}, \mathbf{pk}, \mathbf{apk}, \mathbf{c}, \mathbf{K}, tag) \in \mathsf{List}_{\mathsf{H}_1}$
        if found return $tag$
    $tag \leftarrow \mathsf{Space}_{\mathsf{H}_1}$
    $\boxed{\text{if } \exists\ (*, *, *, *, *, tag) \in \mathsf{List}_{\mathsf{H}_1} \text{ abort}}$ $\mathsf{G}_1$
    add $(fullsid, pk, apk, c, K, tag)$ to $\mathsf{List}_{\mathsf{H}_1}$
    return $tag$

On query $\mathsf{H}_2(fullsid, apk, c, tag, K)$:
    find $(\mathbf{fullsid}, \mathbf{apk}, \mathbf{c}, \mathbf{tag}, \mathbf{K}, key) \in \mathsf{List}_{\mathsf{H}_2}$
        if found return $key$
    $key \leftarrow \mathsf{Space}_{\mathsf{H}_2}$
    add $(fullsid, apk, c, tag, K, key) \in \mathsf{List}_{\mathsf{H}_2}$
    return $key$

On query $\mathsf{H}(fullsid, pw, r)$:
    find $(\mathbf{fullsid}, \mathbf{pw}, \mathbf{r}, R) \in \mathsf{List}_{\mathsf{H}}$
        if found return $R$
    $R \leftarrow \mathsf{Space}_{\mathsf{H}}$
    $\boxed{\text{if } \exists\ (*, *, *, R) \in \mathsf{List}_{\mathsf{H}} \text{ abort}}$ $\mathsf{G}_1$
    add $(fullsid, pw, r, R)$ to $\mathsf{List}_{\mathsf{H}}$
    return $R$

On query $\mathsf{H}'(fullsid, pw, T)$:
    find $(\mathbf{fullsid}, \mathbf{pw}, \mathbf{T}, t) \in \mathsf{List}_{\mathsf{H}'}$
        if found return $t$
    $t \leftarrow \mathsf{Space}_{\mathsf{H}'}$
    $\boxed{\text{if } \exists\ (*, *, *, t) \in \mathsf{List}_{\mathsf{H}'} \text{ abort}}$   $\mathsf{G}_1$
    $\boxed{\text{if } \exists\ (t, *, *, *) \in \mathsf{List}_{\mathsf{IC}} \text{ abort}}$   $\mathsf{G}_3$
    add $(pw, T, t)$ to $\mathsf{List}_{\mathsf{H}'}$
    return $t$
```

```
On query $\mathsf{IC.Enc}(t, r)$:
    find $(t, r, s, mode) \in \mathsf{List}_{\mathsf{IC}}$
        if found return $s$
    $s \leftarrow \mathsf{Space}_{\mathsf{IC}}$
    $\boxed{\text{if } \exists\ (*, *, s, *) \in \mathsf{List}_{\mathsf{IC}} \text{ abort}}$   $\mathsf{G}_2$
    add $(t, r, s, mode = E)$ to $\mathsf{List}_{\mathsf{IC}}$
    return $s$

On query $\mathsf{IC.Dec}(t, s)$:
    find $(\mathbf{t}, r, \mathbf{s}, mode) \in \mathsf{List}_{\mathsf{IC}}$
        if found return $r$
    find $(fullsid, pw, T, \mathbf{t}) \in \mathsf{List}_{\mathsf{H}'}$
        if found
            $(sk, pk) \leftarrow \mathsf{KEM.Keygen}(\lambda)$
            $(r, M) \leftarrow \mathsf{KEM.Split}(pk)$
            $R \leftarrow M \odot T$
            $\boxed{\text{if } (*, *, r, *) \in \mathsf{List}_{\mathsf{H}} \text{ abort}}$   $\mathsf{G}_4$      $\mathsf{G}_6$
            $\boxed{\text{if } (*, *, *, R) \in \mathsf{List}_{\mathsf{H}} \text{ abort}}$   $\mathsf{G}_5$
            add $(fullsid, pw, r, R)$ to $\mathsf{List}_{\mathsf{H}}$
            $apk \leftarrow (s, T)$
            add $(pw, sk, pk, apk)$ to $\mathsf{List}_{\mathsf{secrets}}$
        if not found
            $r \leftarrow \mathsf{Space}_{\mathsf{IC}}$
            $\boxed{\text{if } \exists (t, r, s, *) \in \mathsf{List}_{\mathsf{IC}} \text{ abort}}$   $\mathsf{G}_2$
    add $(t, r, s, mode = D)$ to $\mathsf{List}_{\mathsf{IC}}$
    return $r$
```

```
On query $(\text{NewSession}, sid, P_i, P_j, role)$ from $\mathcal{F}_{\mathsf{PAKE}}$:
    if $role = Bob$
        add $(sid, P_i, P_j, Bob, \perp, \perp)$ to $\mathsf{List}_{\mathsf{transcripts}}$
    if $role = Alice$
        $apk \leftarrow \mathsf{Space}_{apk}$
        $(s, T) \leftarrow apk$
        if $\exists\ (*, *, T, *) \in \mathsf{List}_{\mathsf{H}'}$ abort            $\mathsf{G}_{12}$
        if $\exists\ (*, *, s, *) \in \mathsf{List}_{\mathsf{IC}}$ abort
        add $(\perp, \perp, s, mode = E)$ to $\mathsf{List}_{\mathsf{IC}}$
        add $(sid, P_i, P_j, Alice, msg1, \perp)$ to $\mathsf{List}_{\mathsf{transcripts}}$
        send $apk$ from $P_i$ to $P_j$
    return
```

```
On query $(\text{Send}, msg)$ from $\mathcal{A}$ to $(sid, P_i)$:

    find $(sid, P_i, P_j, role, msg1, msg2) \in \mathsf{List}_{\mathsf{transcripts}}$
        if not found return $\perp$ // ignore query

    // $msg$ is either for Alice or Bob;
    // check consistency of state in the recorded transcript
    if $role = Bob$ && $(msg1 \neq \perp \ || \ msg2 \neq \perp)$ return $\perp$ // ignore query
    if $role = Alice$ && $(msg1 = \perp \ || \ msg2 \neq \perp)$ return $\perp$ // ignore query

    if $role = Bob$
        update record $(sid, P_i, P_j, Bob, msg, \perp) \in \mathsf{List}_{\mathsf{transcripts}}$
        $fullsid \leftarrow (sid, P_j, P_i)$
        // case (a) $msg$ is $apk$ transmitted by Alice,
        // legitimate partner of Bob
        if $\exists (sid, P_j, P_i, Alice, msg, *) \in \mathsf{List}_{\mathsf{transcripts}}$
            $(sk, pk) \leftarrow \mathsf{KEM.Keygen}(\lambda)$            $\mathsf{G}_9$
            $(c, \_) \leftarrow \mathsf{KEM.Encap}(pk)$                        $\mathsf{G}_8$
            $tag \leftarrow \mathsf{H}_1^*(fullsid, apk, c)$
            send $(c, tag)$ to $P_j$
            send $(\text{NewKey}, sid, P_i, \perp)$
            update Bob's record in $\mathsf{List}_{\mathsf{transcripts}}$
            return
        // $apk$ comes from $\mathcal{A}$
        $apk \leftarrow msg1$
        $(s, T) \leftarrow apk$
        for $(\mathbf{fullsid}, pw, \mathbf{T}, t) \in \mathsf{List}_{\mathsf{H}'}$            $\mathsf{G}_{13}$
            find $(\mathbf{t}, r, \mathbf{s}, \mathbf{mode} = \mathbf{E}) \in \mathsf{List}_{\mathsf{IC}}$
```

```
// case (b) $apk$ computed in the forward direction
if record found
    // $pw$ is extractable
    send $(\text{TestPwd}, sid, P_i, pw)$ to $\mathcal{F}_{\mathsf{PAKE}}$ // test $pw$
    if “correct guess"
        // execute the protocol honestly
        $R \leftarrow \mathsf{H}(fullsid, pw, r)$
        $M \leftarrow T \odot R^{-1}$
        $pk \leftarrow \mathsf{KEM.Split}^{-1}(r, M)$
        $(c, K) \leftarrow \mathsf{KEM.Encap}(pk)$
        $tag \leftarrow \mathsf{H}_1(fullsid, pk, apk, c, K)$
        $key \leftarrow \mathsf{H}_2(fullsid, pk, apk, c, K)$
        send $(c, tag)$ to $P_j$                                        $\mathsf{G}_{15}$
        send $(\text{NewKey}, sid, P_i, key)$
        update Bob’s record in $\mathsf{List}_{\mathsf{transcripts}}$
        return
    if “wrong guess"
        // complete session with fresh key
        $(sk, pk) \leftarrow \mathsf{KEM.Keygen}(\lambda)$                 $\mathsf{G}_{16}$
        $(c, K) \leftarrow \mathsf{KEM.Encap}(pk)$
        $tag \leftarrow \mathsf{H}_1^*(fullsid, apk, c)$
        send $(c, tag)$ to $P_j$
        send $(\text{NewKey}, sid, P_i, \perp)$
        update Bob’s record in $\mathsf{List}_{\mathsf{transcripts}}$
        return
// case (c) all other cases
// (e.g. no record of $apk$ in the forward direction)
if no record found
    $(sk, pk) \leftarrow \mathsf{KEM.Keygen}(\lambda)$                     $\mathsf{G}_{14}$
    $(c, \_) \leftarrow \mathsf{KEM.Encap}(pk)$                           $\mathsf{G}_{13}$
    $tag \leftarrow \mathsf{H}_1^*(fullsid, apk, c)$
    send $(c, tag)$ to $P_j$
    send $(\text{NewKey}, sid, P_i, \perp)$
    update Bob’s record in $\mathsf{List}_{\mathsf{transcripts}}$
    return

if $role = Alice$
    update record $(sid, P_i, P_j, Alice, msg1, \mathbf{msg}) \in \mathsf{List}_{\mathsf{transcripts}}$
    $fullsid \leftarrow (sid, P_i, P_j)$
    $(c, tag) \leftarrow msg$
    $apk \leftarrow msg1$
    if $\exists(sid, P_j, P_i, Bob, apk, msg) \in \mathsf{List}_{\mathsf{transcripts}}$
        send $(\text{NewKey}, sid, P_i, \perp)$                           $\mathsf{G}_{8}$
        return
```

```
if $\nexists$ $(\mathit{fullsid}, *, apk, c, *, tag) \in \mathsf{List}_{\mathsf{H}_1}$        $\mathsf{G}_7$
    send $(\text{TestPwd}, sid, P_i, \perp)$
    send $(\text{NewKey}, sid, P_i, \perp)$
    return
// extract $pk$ from $tag$, record must exist        $\mathsf{G}_{10}$
find $(\mathbf{fullsid}, pk, \mathbf{apk}, \mathbf{c}, K, \mathbf{tag})$ to $\mathsf{List}_{\mathsf{H}_1}$
find $(pw, sk, \mathbf{pk}, \mathbf{apk})$ to $\mathsf{List}_{\mathsf{secrets}}$
    if record found        $\mathsf{G}_{11}$
        send $(\text{TestPwd}, sid, P_i, pw)$ to $\mathcal{F}_{\mathsf{PAKE}}$
            if "correct guess"
                // run protocol honestly
                // program the session key
                // and replace placeholders if needed
                $(s, T) \leftarrow apk$
                if $\exists$ $(\perp, \perp, s, mode = E) \in \mathsf{List}_{\mathsf{IC}}$        $\mathsf{G}_{12}$
                    $(r, M) \leftarrow \mathsf{KEM.Split}(pk)$
                    $t \leftarrow \mathsf{H}'(pw, M \odot \mathsf{H}(\mathit{fullsid}, pw, r))$
                    upd. record $(t, r, s, mode = E)$ in $\mathsf{List}_{\mathsf{IC}}$
                $K \leftarrow \mathsf{KEM.Decap}(sk, c)$
                if $tag \neq \mathsf{H}_1(\mathit{fullsid}, pk, apk, c, K)$
                    // send random key via functionality
                    send $(\text{NewKey}, sid, P_i, \perp)$
                    return
                else
                    // tell $\mathcal{F}_{\mathsf{PAKE}}$ to complete session with $key$
                    $key \leftarrow \mathsf{H}_2(\mathit{fullsid}, apk, c, tag, K)$
                    send $(\text{NewKey}, sid, P_i, key)$
                    return
            if "wrong guess"
                // tell $\mathcal{F}_{\mathsf{PAKE}}$ to complete the session
                // with a random key
                send $(\text{NewKey}, sid, P_i, \perp)$
                return
    if no record found        $\mathsf{G}_{10}$
        send $(\text{TestPwd}, sid, P_i, \perp)$
        send $(\text{NewKey}, sid, P_i, \perp)$
        return
```

B IND-CCA $\Longrightarrow$ OW-PCA

A plaintext-checking oracle is obviously less powerful than a full-fledged decapsulation oracle. However, in IND-CCA security experiment (refer to Fig. 5 below), the adversary is prohibited from calling the decapsulation oracle on the challenge ciphertext, whereas in the OW-PCA game (Fig. 1), there are no such restrictions on the adversary's queries. In fact, for the tighter reduction shown in the proof of Theorem 1, we cannot impose such a restriction because the reduction must check the challenge ciphertext against the key provided by the adversary (games G_8, G_{13}, and G_{15}). Thus, the reader might wonder if an IND-CCA-secure KEM, which is the standard notion of security for KEMs, actually implies that the KEM is OW-PCA-secure with a tight reduction. We answer this question affirmatively here.

We also note that we *do* forbid the adversary in ANO-PCA game (Fig. 1) from querying the challenge ciphertext. This is because the reduction only needs to decapsulate adversarially-crafted ciphertexts, which are different from the challenge ciphertext (games G_9, G_{14}, and G_{16}).

Definition 6. ***(KEM indistinguishability against chosen ciphertext attacks)*** *A Key Encapsulation Mechanism (KEM) scheme is said to be* IND-CCA *secure if for any PPT adversary $\mathcal{A}$ engaged in the* IND-CCA *security game, where $\mathcal{A}$ is prohibited from calling the* Decap *oracle on the challenge ciphertext, the advantage of $\mathcal{A}$ defined as:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM},\mathcal{A}}(\lambda) \overset{\text{def}}{=} 2 \cdot \Pr[\mathsf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{KEM}}(\lambda) = 1] - 1 \tag{27}$$

is a negligible function of the security parameter λ. Experiment IND-CCA *is defined in Fig. 5.*

```
Exp $\mathsf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{KEM}}(\lambda)$
    $(pk, sk) \leftarrow \mathsf{Keygen}(\lambda)$
    $(c^*, K_0) \leftarrow \mathsf{Encap}(pk)$
    $K_1 \leftarrow \mathcal{K}$
    $b \leftarrow\$ \{0,1\}$
    $b' \leftarrow \mathcal{A}^{\mathsf{Decap}_{c^*}(\mathsf{sk},\cdot)}(pk, c^*, K_b)$
    $\mathbf{return}\ b == b'$

Oracle $\mathsf{Decap}_{c^*}(\mathsf{sk}, c)$
    if $c == c^* \mathbf{return}\ \perp$
    $K \leftarrow \mathsf{Decap}(\mathsf{sk}, c)$
    $\mathbf{return}\ K$
```

Fig. 5. Security experiment defining indistinguishability against chosen ciphertext attacks (IND-CCA). $\mathcal{A}$ is prohibited from calling the Decap oracle on the challenge ciphertext c^*.

Lemma 2. *If* KEM *is a* IND-CCA *secure key encapsulation mechanism, then it is also* OW-PCA *secure.*

Proof. Let $\mathcal{A}$ be any adversary against game OW-PCA. We construct an adversary $\mathcal{B}$ against IND-CCA that simulates game OW-PCA for $\mathcal{A}$ as follows: $\mathcal{B}$ receives (pk, c^*, K^*) from the IND-CCA challenger and runs $\mathcal{A}(pk, c^*)$. $\mathcal{B}$ has to provide a PCO oracle to $\mathcal{A}$ and has access to a Decap oracle that can be queried on anything except c^*.

Here's how $\mathcal{B}$ answers PCO queries:

- If $\mathcal{A}$ queries $\mathsf{PCO}(c, K)$ for $c \neq c^*$, $\mathcal{B}$ calls $\mathsf{Decap}(c)$ and checks the result against K.
- If $\mathcal{A}$ queries $\mathsf{PCO}(c^*, K^*)$, $\mathcal{B}$ guesses $b = 0$ (i.e. ciphertext c^* encrypts K^*) and terminates. Note that K^* is high entropy and is perfectly hidden from $\mathcal{A}$ unless c^* encrypts it, and the probability that $\mathcal{A}$ queries $\mathsf{PCO}(c^*, K^*)$ without c^* actually encrypting K^* is statistically negligible.
- If $\mathcal{A}$ queries $\mathsf{PCO}(c^*, K)$, for $K \neq K^*$, $\mathcal{B}$ responds with false (indicating that c^* does not encrypt K).

When $\mathcal{A}$ terminates and outputs K', $\mathcal{B}$ guesses $b = 0$ if $K' == K^*$ (same reason as to why $\mathcal{B}$ terminates on query $\mathsf{PCO}(c^*, K^*)$), otherwise $\mathcal{B}$ picks a random b as its final guess for the $\mathsf{IND\text{-}CCA}$ game.

The simulation of $\mathsf{OW\text{-}PCA}$ game is perfect, unless $b = 1$ and $\mathcal{A}$ queried $\mathsf{PCO}(c^*, K_b)$, meaning that at least half of the time, a victory of $\mathcal{A}$ against $\mathsf{OW\text{-}PCA}$ translates to a victory of $\mathcal{B}$ against $\mathsf{IND\text{-}CCA}$.

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM},\mathcal{B}}(\lambda) &\overset{\text{def}}{=} 2 \cdot \Pr[\mathsf{IND\text{-}CCA}^{\mathcal{B}}_{\mathsf{KEM}}(\lambda) = 1] - 1 && (28)\\
&= 2 \cdot \Pr[b = b'] - 1 && (29)\\
&= 2 \cdot (\Pr[b = 0 \wedge b' = 0] + \Pr[b = 1 \wedge b' = 1]) - 1 && (30)\\
&= \Pr[b' = 0 | b = 0] + \Pr[b' = 1 | b = 1] - 1 && (31)\\
&= \Pr[b' = 0 | b = 0] + (\frac{1}{2} - \epsilon) - 1 && (32)\\
&= \Pr[K' = K^*] + \frac{1}{2} \cdot \Pr[K' \neq K^*] + (\frac{1}{2} - \epsilon) - 1 && (33)\\
&= \mathsf{Adv}^{\mathsf{OW\text{-}PCA}}_{\mathsf{KEM},\mathcal{A}}(\lambda) + \frac{1}{2} \cdot (1 - \mathsf{Adv}^{\mathsf{OW\text{-}PCA}}_{\mathsf{KEM},\mathcal{A}}(\lambda)) + (\frac{1}{2} - \epsilon) - 1 && (34)\\
&= \frac{1}{2} \cdot \mathsf{Adv}^{\mathsf{OW\text{-}PCA}}_{\mathsf{KEM},\mathcal{A}}(\lambda) - \epsilon && (35)
\end{aligned}$$

The term $\epsilon = \frac{n}{|\mathcal{K}|}$ is statistically negligible and depends on the number n of queries that $\mathcal{A}$ makes to the PCO oracle and the size of the encapsulation key space $\mathcal{K}$.

□

C Kyber has splittable and pseudo-uniform public keys

Theorem 2. Kyber *has splittable and pseudo-uniform public keys.*

Proof. Kyber works over ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, where $q = 3329$ is a small prime and $n = 256$. A public key consists of two parts:

1. A seed $\rho \in \{0,1\}^{256}$ that is derived from expanding a purely random bitstring $d \in \mathbb{B}^{32}$ using a hash function $G(d)$ that produces two 32-byte outputs, with ρ being one of them.
2. A byte encoding of a vector $\mathbf{t} \in R_q^k$, where $k \in \{2,3,4\}$ is fixed by the security parameter λ.

Let $\mathsf{Split} : \mathcal{PK}_\lambda \to N_\lambda \times G_\lambda$ be the function that trivially decomposes Kyber public keys into ρ and $\mathbf{t}$, and let Split^{-1} be the reverse operation. This fulfills Condition (1) of Definition 4.

In the security proofs of Kyber [28,15,20,29], function G is modeled as a random oracle, which ensures that the distribution of ρ is uniform. ρ is then further expanded into a large stream of candidate 12-bit values via an eXtendable Output Function (XOF). From this stream, the first candidate values within the range $[0, 3329)$ are selected to form the public matrix $\mathbf{A} \in R_q^{k \times k}$ in the NTT domain. This process is known as *rejection sampling.*

More precisely, each entry in matrix $\mathbf{A}$ is sampled independently by calling a procedure Parse that takes as input the seed ρ and some public domain separation inputs, including column and row indices. Parse first uses its input to seed the XOF, which Kyber instantiates with SHAKE-128. The procedure then rejection-samples one coefficient at a time, by taking the next 12 bits from the output of SHAKE-128 and checking if they encode an integer in the range $[0, q)$. Values that are out of range are discarded, so the polynomial is intuitively just the first set of 12-bit sequences produced by SHAKE-128 that fall within the correct range.

Public matrix $\mathbf{A}$ and vector $\mathbf{t} = \mathbf{A}^T s + e$, where s (the secret key) and e the ephemeral noise are sampled from a suitable (low norm) distribution, form a Module-LWE instance. Therefore, modeling G and the procedure that maps ρ to $\mathbf{A}$ as random oracles, ensures that Kyber public keys are pseudo-uniform (security experiment UNI-PK in Fig. 1) under the decisional MLWE assumption in the Random Oracle Model. This satisfies Condition (3) of Definition 4.

It remains to demonstrate that Condition (2) is also satisfied. More precisely, using the terminology of both the definition and the construction of CHIC in Fig. 4, we need to show that there exists an RO-indifferentiable hash function H that maps elements from $N_\lambda = \{0,1\}^{256}$ to $G_\lambda = R_q^k$.[18]

To implement H, we simply repurpose the rejection sampling procedure used in Kyber key generation for sampling matrix A. However, instead of rejection-sampling $k \times k$ elements in R_q, one coefficient at the time, we sample only k elements to match the size of vector $\mathbf{t}$. We first note that, because each element in R_q is sampled independently using a domain-separated instance of SHAKE-128, it suffices to prove that Parse is indifferentiable from a random oracle that maps a seed ρ to a polynomial in R_q. Let $\mathcal{O}$ denote SHAKE-128 and let $\mathcal{I}$ denote the ideal random function that maps each seed ρ to a uniform value in R_q.

[18] Note that this proof of indifferentiability also justifies why the rejection sampling procedure that maps ρ to $\mathbf{A}$ can be seen as a random oracle when justifying the pseudorandomness of $\mathbf{t}$ under decisional MLWE.

Indifferentiability requires that a simulator $\mathcal{S}$ can explain the outputs of Parse to an adversary with access to $\mathcal{O}$ in the following sense.

$$\mathcal{A}^{\mathsf{Parse}^{\mathcal{O}}(\cdot),\mathcal{O}(\cdot)} \sim \mathcal{A}^{\mathcal{I}(\cdot),\mathcal{S}^{\mathcal{I}}(\cdot)} .$$

We now describe the simulator $\mathcal{S}$. When $\mathcal{A}$ queries $\mathcal{O}$ on fresh input ρ, $\mathcal{S}$ can obtain a ring element $y \in R_q$ by querying $\mathcal{I}(\rho)$—recall that Parse just passes its own input to the XOF, so $\mathcal{S}$ immediately knows what the adversary sees at the output of $\mathcal{I}$ for that specific seed. Then, $\mathcal{S}$ runs $\mathsf{Parse}^{\mathcal{O}}(\rho)$ lazily sampling $\mathcal{O}$ as needed, to obtain a discardable ring element y'. This creates a state of $\mathcal{O}$ that is consistent with a fresh execution of Parse for ρ, and this has never been seen by the adversary. One characteristic of this trace is that $\mathcal{S}$ can easily identify the values that gave rise to coefficients in y': these are sequences of 12-bits that contain those coefficients, and they correspond to the first $n = 256$ positions in the trace that yield values in the range $[0, q)$. Now $\mathcal{S}$ simply rewrites this trace replacing the y' coefficients with y coefficients. The resulting trace is a perfect simulation of a trace that produces y. This means that all future queries to $\mathcal{O}$ associated with ρ will be consistent with the value of y observed by the adversary in the output of $\mathcal{I}$, exactly as it occurs in the real world. This simulation is, in fact, perfect and it follows that Parse is indifferentiable from $\mathcal{I}$ as required.

□