GeT a CAKE: Generic Transformations from Key Encaspulation Mechanisms to Password Authenticated Key Exchanges

Hugo Beguinet[1][2], Céline Chevalier[1][3], David Pointcheval[1], Thomas Ricosset[2], and Mélissa Rossi[4]

[1] DIENS, École Normale Supérieure, CNRS, Inria, PSL University, Paris, France,
hugo.beguinet, celine.chevalier, david.pointcheval@ens.fr
[2] Thales, Gennevilliers, France,
hugo.beguinet, thomas.ricosset@thalesgroup.com
[3] CRED, Université Paris-Panthéon-Assas, Paris, France
[4] ANSSI, Paris, France,
melissa.rossi@ssi.gouv.fr

Abstract. Password Authenticated Key Exchange (PAKE) have become a key building block in many security products as they provide interesting efficiency/security trade-offs. Indeed, a PAKE allows to dispense with the heavy public key infrastructures and its efficiency and portability make it well suited for applications such as Internet of Things or e-passports. With the emerging quantum threat and the effervescent development of post-quantum public key algorithms in the last five years, one would wonder how to modify existing password authenticated key exchange protocols that currently rely on Diffie-Hellman problems in order to include newly introduced and soon-to-be-standardized post-quantum key encapsulation mechanisms (KEM). A generic solution is desirable for maintaining modularity and adaptability with the many post-quantum KEM that have been introduced.
In this paper, we propose two new generic and natural constructions proven in the Universal Composability (UC) model to transform, in a black-box manner, a KEM into a PAKE with very limited performance overhead: one or two extra symmetric encryptions. Behind the simplicity of the designs, establishing security proofs in the UC model is actually non-trivial and requires some additional properties on the underlying KEM like fuzziness and anonymity. Luckily, post-quantum KEM protocols often enjoy these two extra properties. As a demonstration, we prove that it is possible to apply our transformations to Crystals-Kyber, a lattice-based post-quantum KEM that will soon be standardized by the National Institute of Standards and Technology (NIST).
In a nutshell, this work opens up the possibility to securely include post-quantum cryptography in PAKE-based real-world protocols.

Keywords: Key Encapsulation Mechanism · Password-Authenticated Key Exchange · Universal Composability

1 Introduction

A Password Authenticated Key Exchange (PAKE) protocol allows two users to derive a secret key over insecure channels only with the premise of sharing the same low entropy password. PAKE has become increasingly relevant in recent years due to the proliferation of connected devices and the growing demand for secure communication in scenarios where a public key infrastructure (PKI) may not be practical or desirable. It is particularly appealing for use cases like the Internet of Things (IoT) or e-passports, where portability, independence, and efficiency are important considerations. For IoT devices, for example, PKI is not feasible because of the number of devices and their limited computing resources and connectivity. PAKE allows these devices to securely communicate with each other using a simple password that can be easily changed if compromised. Similarly, in the case of e-passport, PAKE can be used to establish secure communication between the passport and a reader without the need for a PKI. It allows for a more portable and independent solution, as no central authority is necessary to verify a passport's authenticity. Overall, PAKE offers a trade-off between security and efficiency as compared to traditional authenticated key exchange protocols in certain circumstances.

Security models for PAKEs The security of PAKE will always be weaker than the security of PKI-based authenticated key exchange. Indeed, the presence of a low-entropy password allows powerful dictionary attacks. The conceptual idea in the PAKE security models is to accept the possibility of such dictionary attacks *but to prove that they must be made online*, i.e. that no password validity test is accessible offline. This slight security regression compared to authenticated key exchange is often accepted because online dictionary attacks are rarely relevant in practical contexts and the efficiency gain of PAKE is much higher. Moreover one can always block a user after a certain number of failed attempts. More formally, for proving the security of PAKE, the dictionary attacks should then be materialized in the existing security models for authenticated key exchange. Several solutions have emerged and have been refined over the last decade. Today, there are two main security models for PAKE protocols: the Bellare-Pointcheval-Rogaway [BPR00] and the Universal Composability (UC) model [Can01,CR03] with its PAKE's version [CHK+05]. The BPR model, introduced by Bellare, Pointcheval, and Rogaway, is a game-based security model that uses specific games to evaluate the ability of an adversary to break the protocol. On the other hand, the UC model, introduced by Canetti, is a simulation-based model that provides strictly better security guarantees, as stated in the original UC PAKE paper [CHK+05].

Existing work on PAKE. The concept of PAKE was formalized and analyzed during the 1990s by Bellovin and Meritt with the Encrypted Key Exchange (EKE) protocol [BM92]. Since then, various PAKE protocols have been proposed, with some standardized by organizations such as the Internet Engineering Task Force (IETF) [Sch17]. Over the years, two main categories of PAKE appeared.

The first use passwords to obscure the exchanged messages while the second use them as part of the randomness to build the necessary material, like the group generator. EKE [BM92] and OEKE [BCP03] are typical examples of the former. And SPEKE [Mac01] or CPace [AHH22] are examples for the latter. While many different PAKE designs have been introduced, not all are proven in the strong UC security model. In the previous examples of PAKE constructions, EKE [DHP+18], OEKE [ACCP08] and CPace [AHH21] have been proven in the UC model.

Post-quantum threat. While vastly used in current security products like IoT or e-passports, all these PAKE constructions rely on the Diffie-Hellman key exchange to provide cryptographic security. It raises concerns about their long-term security, as the emergence of quantum computing in recent years threatens any Diffie-Hellman-based key exchange, and thus any currently used PAKE. Indeed, quantum computers would potentially break, even retroactively, the mathematical foundations of many current cryptographic systems including the difficulty of the Diffie-Hellman problem. Therefore, it is crucial to carefully consider the long-term security of PAKE protocols and design them accordingly. In response to this potential threat to current cryptographic systems, the National Institute of Standards and Technology (NIST) has launched a standardization process for post-quantum cryptographic primitives in 2017. The goal of this campaign is to provide new post-quantum standards for two basic and crucial cryptographic building blocks: Key Encapsulation Mechanisms (KEMs) and digital signatures. These two families of public-key algorithms may be used on their own but more importantly, the future standards are destined to be included as black-boxes in internet and IoT protocols to complement the pre-quantum bricks. Many different families of mathematical problems were used for the design of candidate algorithms like error correcting codes or lattices. The analysis of the different candidate algorithms is currently ongoing but the NIST has announced a first set of standards in 2022 including the lattice-based KEM Crystals-`Kyber` [SAB+22]. More recently, specific PAKE constructions using post-quantum cryptography, in particular, lattices assumptions, were introduced. However most lattice constructions are either proven in weaker security models [GDLL17] or using mechanisms that are highly inefficient in practice [BCV19,ZY17].

1.1 Our Contributions

This paper proposes the first generic constructions to transform a black-box KEM into a PAKE. The idea is natural and inspired from EKE and OEKE. In high level, it consists in encrypting the public key using the password as a secret key. A second modification consists in either encrypting the ciphertext with that same password or adding an authentication tag. The first transformation is called CAKE, derived from K(EM-to-P)AKE, and the second transformation is called OCAKE. Both constructions are graphically sketched later in the paper in Figures 5 and 6. By design, they are simple, efficient and easy to implement. However, the price for such simplicity must be paid on the analysis side.

Let us first intuitively discuss the requirements on the KEM for achieving formal security. Consider a KEM where the public key is designed with a particular shape, for example, the public key might always be composed of small coefficients. When encrypted, the distribution of the sent message would look uniformly distributed. However, any attacker may perform an offline dictionary attack: given an encrypted public key, it will be possible to leverage the particular public key form as a condition for the valid password. In such case, the correct password will be the one that decrypts to a public key with small coefficients. Hence, an indistinguishability property on the distribution of the public key, called *fuzziness* (formally defined in Definition 3), will be required to avoid offline dictionary attacks. Likewise, such property on the ciphertext, called *anonymity* (formally defined in Definition 4), will be essential. In addition, another important property should be fulfilled by the symmectric encryption to construct such PAKE. The needed property is ensured by the Ideal Cipher (IC) model [BPR00]. It consists in assuming that the encryption behaves like a random permutation on every key. While it does not retain clear weaknesses to use a relaxed model, an ideal cipher is necessary to unwrap the proofs of our theorems.
In this paper, we successfully prove our CAKE and OCAKE constructions in the UC model assuming the above properties, the random oracle model (ROM) and the erasure model stating that any obsolete internal information is erased.

Why two constructions? Similarly to EKE and OEKE, CAKE and OCAKE offer slightly different security/efficiency trade-offs. Let us compare both constructions:

- The first construction, CAKE, consists in encrypting both exchanged messages. It leads to an implicitly authenticated key exchange protocol based on passwords. However a participant is not sure that the opposing party is able to obtain the session key. This assurance can only be achieved by explicit authentication. The security model for proving CAKE is very strong as it captures adaptive corruptions. In other words, the model allows an attacker to corrupt a user and thus obtain all its internal state in an adaptive way during an ongoing execution of the protocol.
- In order to add explicit authentication of the receiver, one usually includes a key-confirmation tag. In this case, one can remark with OCAKE that only one symmetric encryption is required. The second encryption is just replaced by the authentication that provides an explicit authentication to the receiver. However, this construction can only be proven secure in the static corruption model where the attacker may still corrupt users but the choice should be made before the execution of the protocol. Additionally, it is here possible to add an explicit client authentication at the end of the exchange to provide mutual explicit authentication.

In complement, we propose to show that the assumed properties on the KEM are not just artifacts that allow our proof to work. They are actually verified in concrete KEMs. We choose the example of Crystals-`Kyber` [SAB$^+$22] as our guinea pig for applying our transformations. Crystals-`Kyber` is a future NIST post-quantum standard. We formally demonstrate that `Kyber` validates fuzziness and anonymity leading up to security statements for CAKE-Kyber (Theorem 3) and OCAKE-Kyber (Theorem 4).

1.2 Outline of the paper

In Section 2, we introduce the preliminary notions on KEM, their security properties with some lattice definitions and a brief introduction to `Kyber`. In Section 3, we provide all the necessary information about PAKEs and their security in the UC model. In Section 4, we present both our KEM to PAKE transformations along with their security statements. For space reasons, the proofs are sketched in the main body of the paper and the full proofs are detailed in Appendices A and B. Finally, we demonstrate our techniques on `Kyber` in Section 5.

2 Preliminaries

2.1 Notations

We note scalars, vectors, and matrices with lowercase plain (i.e. n), lowercase bold (i.e. $\mathbf{e}$), and uppercase bold (i.e. $\mathbf{A}$), respectively. We denote by $\mathsf{negl}(\kappa)$ a negligible function of a security parameter κ. Given a finite set S, the notation $x \leftarrow\!\!\$\, S$ means a uniformly random assignment of an element of S to the variable x. We note KEM the denomination of a key exchange mechanism and refer to `KEM` for the specific key encapsulation mechanism algorithm.

2.2 Key Encapsulation Mechanism

Even if the Key Encapsulation Mechanism's denomination is relatively recent, KEMs have been widely used throughout the history of public key cryptography. The first illustration is the fact that ElGamal [ElG85], based on the Diffie-Hellman key exchange, can be easily seen as a KEM. We will demonstrate later in this section that it enjoys several security notions, such as *semantic security*, *fuzziness*, and *anonymity*. Thereafter, with the NIST competition, many new researches have conducted to the introduction of KEM using a wide variety of structures. Let us cite a few examples: SABER [DKRV18,DKR+20], Crystals-`Kyber` [BDK+18,SAB+22], NewHope [ADPS16,PAA+19] on lattice-based assumptions or alternatively McEliece[McE78,ABC+22] on code-based assumptions.

Definition 1 (Key Encapsulation Mechanism). *A Key Encapsulation Mechanism (*`KEM`*) is a triple of algorithms (*`KeyGen`*,* `Encaps`*,* `Decaps`*):*

- `KeyGen`*: Returns a of pair public-secret keys* $(pk, sk) \in \mathcal{P} \times \mathcal{SK}$
- `Encaps`*: Takes a public key* $pk \in \mathcal{P}$ *as input to produce a ciphertext* $c \in \mathcal{C}$ *and a key* $K \in \mathcal{K}$*. The ciphertext* c *is called an encapsulation of the key* K*;*

– $\texttt{Decaps}$*: Takes a secret key* $sk \in \mathcal{SK}$ *and an encapsulation* $c \in \mathcal{C}$ *as input, and outputs* $K \in \mathcal{K}$*.*

where $\mathcal{SK}$*,* $\mathcal{P}$*,* $\mathcal{C}$*, and* $\mathcal{K}$ *are the sets of secret keys, public keys, ciphertexts and session keys.*

The formalization of the sets $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{K}$ will impact the security notions presented in the sequel.

Correctness. The correctness of a KEM requires that, for a security parameter κ,

$$\Pr\left[\begin{array}{l}(pk, sk) \leftarrow\!\!\$\ \texttt{KeyGen}(1^\kappa) \\ (c, K) \leftarrow \texttt{Encaps}(pk)\end{array} : \texttt{Decaps}(sk, c) = K\right] > 1 - \mathsf{negl}(\kappa).$$

Security Notions. The usual security notion for KEM is *semantic security*, also known as *indistinguishability*:

Definition 2 (Indistinguishability). *We define the advantage of any adversary* $\mathcal{A}$ *in deciding the key of the* $\texttt{KEM}$ *by:*

$$\mathsf{Adv}^{\text{ind}}_{\texttt{KEM}}(\mathcal{A}) = \left| \Pr_{\mathsf{D}_R}[\mathcal{A}(c, K) = 1] - \Pr_{\mathsf{D}_\$}[\mathcal{A}(c, K') = 1] \right|.$$

where we consider the real and random distributions

$$\begin{aligned} \mathsf{D}_R &= \{(pk, sk) \leftarrow \textit{KeyGen}(1^\kappa); (c, K) \leftarrow \textit{Encaps}(pk) : (c, K)\}, \\ \mathsf{D}_\$ &= \{(pk, sk) \leftarrow \textit{KeyGen}(1^\kappa); (c, K) \leftarrow \textit{Encaps}(pk); K' \leftarrow\!\!\$\ \mathcal{K} : (c, K')\}. \end{aligned}$$

In all the advantage definitions, we will denote $\mathsf{Adv}^{\text{ind}}_{\texttt{KEM}}(t)$ the maximal advantage any adversary can have within time t.
Let us introduce additional properties for KEMs, on the distributions of the public keys and of the encapsulations. We will denote by *fuzziness* the randomness of public keys, and by *anonymity* the randomness of the encapsulation. The latter is the usual definition, when the ciphertext distribution does not depend on the public key, and thus does not leak any information about the recipient.

Definition 3 (Fuzzy KEM). *A* $\texttt{KEM}$ *is said fuzzy if the distribution of the public keys output by the* $\textit{KeyGen}$ *algorithm are computationally indistinguishable from uniform keys in* $\mathcal{P}$*. More formally, we define the advantage of any adversary* $\mathcal{A}$ *in breaking the fuzziness of the* $\texttt{KEM}$ *by:*

$$\mathsf{Adv}^{\text{fuzzy}}_{\texttt{KEM}}(\mathcal{A}) = \left| \Pr_{\mathsf{D}_R}[\mathcal{A}(pk) = 1] - \Pr_{\mathsf{D}_\$}[\mathcal{A}(pk) = 1] \right|,$$

where

$$\mathsf{D}_R = \{(pk, sk) \leftarrow \textit{KeyGen}(1^\kappa) : pk\} \qquad \textit{and} \qquad \mathsf{D}_\$ = \{pk \leftarrow\!\!\$\ \mathcal{P} : pk\}.$$

Definition 4 (Anonymous KEM). *A* KEM *is said anonymous if the distribution of the ciphertexts outputted by the* **Encaps** *algorithm are computationally indistinguishable from uniform ciphertexts in $\mathcal{C}$. More formally, we define the advantage of any adversary $\mathcal{A}$ in breaking the anonymity of the* KEM *by:*

$$\mathsf{Adv}^{\mathrm{ano}}_{\mathtt{KEM}}(\mathcal{A}) = \left| \Pr_{\mathsf{D}_R}[\mathcal{A}(c) = 1] - \Pr_{\mathsf{D}_\$}[\mathcal{A}(c) = 1] \right|,$$

where

$$\mathsf{D}_R = \{(pk, sk) \leftarrow \mathtt{KeyGen}(1^\kappa); (c, K) \leftarrow \mathtt{Encaps}(pk) : c\} \text{ and}$$
$$\mathsf{D}_\$ = \{c \leftarrow\!\!\$\, \mathcal{C} : c\}.$$

ElGamal Key Encapsulation Mechanism. In order to illustrate the above notions, let us consider the particular KEM derived from the so-called ElGamal encryption scheme [ElG85]. Let $\mathbb{G}$ be a group of prime order q, spanned by an element g:

- KeyGen(1^κ): chooses a random $x \leftarrow\!\!\$\, \mathbb{Z}_q$ and sets $sk \leftarrow x$, $pk \leftarrow g^x$, with $\mathcal{SK} = \mathbb{Z}_q$ and $\mathcal{P} = \mathbb{G}$;
- Encaps(pk): chooses a random $r \leftarrow\!\!\$\, \mathbb{Z}_q$ and sets $c \leftarrow g^r$, $K \leftarrow pk^r$, with $\mathcal{C} = \mathbb{G}$ and $\mathcal{K} = \mathbb{G}$;
- Decaps(sk, c): outputs $K \leftarrow c^{sk}$.

It is well-know that the indistinguishability of this KEM relies on the Decisional Diffie-Hellman assumption. From the above description, this is clear that public keys are uniformly distributed in $\mathbb{G}$, hence this KEM is fuzzy; and the ciphertexts are also uniformly distributed in $\mathbb{G}$, thus this KEM is also anonymous. Note that the ElGamal KEM actually validates even stronger properties: perfect fuzziness (or smoothness) and perfect anonymity. The perfect nature comes from the fact that these notions are no longer computational but statistically ensured. As will be later stated in Remark 1, these properties are less common for post-quantum KEM protocols and thus will not be considered as requirements for our constructions.

2.3 Learning with Errors

Three rounds of the NIST standardization campaign are already over and one type of hardness assumption seems to be more enticing: lattices. Lattice problems provide strong worst-case to average-case reductions, making them excellent candidates for long-term security. Indeed state of the art algorithm using quantum adversaries are not far more efficient compared to current existing algorithm to solve LWE. In this section, we introduce the Learning With Errors (LWE) [Reg06] assumptions. It can be divided into two problems: a decisional and a search problems. Both are assumed intractable in reasonable time, even for a quantum computer. Let us introduce the decisional version.

We directly consider this problem in a module structure named Module-LWE (we refer to [LS15] for more details). We define $\mathcal{R}_q$ as the ring $\mathbb{Z}_q[X]/(X^n+1)$. Let β_η be the distribution on $\mathcal{R}_q$ where each coefficient of the polynomial is generated according to a centered binomial distribution with parameter 2η. We define the oracle $\mathcal{O}^{\mathsf{mlwe}}_{m,k,\eta}$ that outputs samples of the form $(\mathbf{A}, \mathbf{b} = \mathbf{A}\cdot\mathbf{s}+\mathbf{e})$ with $\mathbf{s} \leftarrow\$ \beta_\eta^k$, $\mathbf{A} \leftarrow\$ \mathcal{R}_q^{m\times k}$, and $\mathbf{e} \leftarrow\$ \beta_\eta^m$.

Definition 5 (Decisional MLWE$_{m,k,\eta}$ Problem). *Given a set of parameters $m, k, \eta \in \mathbb{N}$, the advantage of any probabilistic polynomial time algorithm $\mathcal{A}$ in deciding the* **d-MLWE** *over $\mathcal{R}_q$ is:*

$$\mathsf{Adv}^{\mathsf{d-mlwe}}_{m,k,\eta}(\mathcal{A}) = \left| \begin{array}{l} \Pr[(\mathbf{A},\mathbf{b}) \leftarrow \mathcal{O}^{\mathsf{mlwe}}_{m,k,\eta} : \mathcal{A}(\mathbf{A},\mathbf{b}) = 1] \\ -\Pr[(\mathbf{A},\mathbf{b}) \leftarrow\$ \mathcal{R}_q^{m\times k} \times \mathcal{R}_q^m : \mathcal{A}(\mathbf{A},\mathbf{b}) = 1] \end{array} \right| .$$

2.4 CRYSTALS-Kyber

Crystals-`Kyber`, also known as `Kyber`, is a Module-LWE-based `KEM` that is one of the most efficient post-quantum solutions. It was introduced in response to the NIST call for standardization of post-quantum primitives and was accepted as the first post-quantum standard for key exchange in 2022. In its original paper [BDK$^+$18], `Kyber` is proposed as a `KEM` that is secure against chosen-plaintext attacks (CPA-secure) and then achieves chosen-ciphertext attacks (CCA-secure) with the Fujisaki-Okamoto transform.
In Figure 1, we present the CPA-secure version of `Kyber`, where $\mathcal{R}_q$ is the ring $\mathbb{Z}_q[X]/(X^n+1)$. Following the last supplemented version (3.0) [SAB$^+$22], the suggested parameters are defined as follows: (X^n+1) is the $2n$-th cyclotomic polynomial where n and q are equal respectively to 256 and $q = 3329$.
Additionally, for the sake of efficiency, `Kyber` includes an optimization using a compression function that can be thought of as a bit cut. While we do not use this compression in our protocol for the sake of clarity, it should be included in any implementation of our protocols using `Kyber` for maximum efficiency and correctness. We refer to the most recent NIST submission package [SAB$^+$22] for more detailed information.

3 Password Authenticated Key Exchange

3.1 Introduction to PAKE

Initially introduced by Bellovin and Merritt [BM92], a Password-Authenticated Key Exchange (PAKE) is a protocol that allows two parties to establish a shared secret session key over an insecure communication channel using a password as the only authentication means. The goal of PAKEs is to ensure that the key exchange is secure even if the password is weak or stolen by an attacker, the only possible attack being an online exhaustive search, which can be detected and stopped using some organizational action.

```
Kyber.KeyGen($1^\kappa$)
1 : $\rho, \sigma \leftarrow\$ \{0,1\}^\kappa$
2 : $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times k} := \mathtt{Sam}(\rho)$
3 : $(\mathbf{s}, \mathbf{e}) \leftarrow \beta_\eta^k \times \beta_\eta^k := \mathtt{Sam}(\sigma)$
4 : $\mathbf{b} \leftarrow \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$
5 : return $(pk = (\rho, \mathbf{b}), sk = \mathbf{s})$
```

```
Kyber.Encaps($pk = (\rho, \mathbf{b})$)
1 : $\tau \leftarrow\$ \{0,1\}^\kappa$
2 : $m \leftarrow\$ \{0,1\}^n \subseteq \mathcal{R}_q$
3 : $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times k} := \mathtt{Sam}(\rho)$
4 : $(\mathbf{r}, \mathbf{e}', e") \leftarrow \beta_\eta^k \times \beta_\eta^k \times \beta_\eta := \mathtt{Sam}(\tau)$
5 : $\mathbf{u} \leftarrow \mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}'$
6 : $v \leftarrow \mathbf{b}^T \cdot \mathbf{r} + e" + \left\lceil \frac{q}{2} \right\rfloor \cdot m$
7 : return $c \leftarrow (\mathbf{u}, v)$
```

```
Kyber.Decaps($sk = \mathbf{s}, c = (\mathbf{u}, v)$)
1 : return $\left\lceil \frac{2}{q} \left( v - \mathbf{s}^T \cdot \mathbf{u} \right) \right\rfloor$
```

Fig. 1. Simplified Kyber KEM: Kyber.KeyGen, Kyber.Encaps, Kyber.Decaps

PAKE protocols are handy when strong authentication is required while other forms of authentication (certificates) are not usable. They are often used in combination with other protocols to provide a secure channel for communication.

PAKE protocols might be vulnerable to two types of attacks: offline-dictionary attacks and online-dictionary attacks. The former occurs when an attacker gains knowledge of the password using pre computed lists of common passwords and exchanged information. Whereas, the latter involves an attacker actively trying to obtain the password by attempting to log in with different guesses. PAKE protocols often implement measures such as limiting the number of tries an attacker can make to guess the password to protect against online-dictionary attacks. Consequently, the security of a PAKE protocol ultimately relies on its resistance to offline-dictionary attacks. In other words, the strength of a PAKE protocol is determined by how difficult it is for an attacker to guess the password from the public transcript, even if it has a lof of time and resources.

3.2 The Universal Composability (UC) Model

Overview of the UC Framework. The Universal Composability (UC) model [Can01] is a simulation-based model in which an environment $\mathcal{Z}$ attempts to differentiate the output of a protocol execution Π in the real world from the output generated in an ideal world. In the real world, the execution takes place between parties and an potential adversary. In the ideal world, dummy players and an ideal adversary or simulator $\mathcal{S}$ interact solely with an ideal functionality $\mathcal{F}$ to compute a specific function f. The ideal functionality can be informally defined as a trusted party that honestly and unconditionally responds to any query. A schematic representation is given in Figure 2.
The original paper [Can01] only uses sid as session identifiers, but a improved

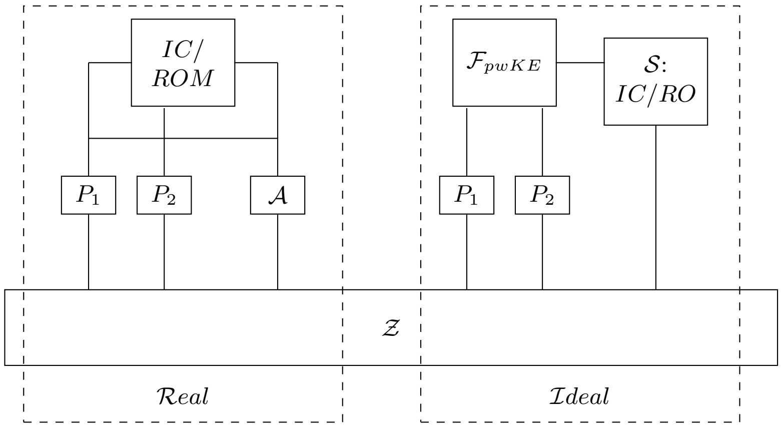

Fig. 2. $\mathcal{R}eal$ versus $\mathcal{I}deal$ world: $\mathcal{Z}$ capability.

version of the UC model was published in [CR03] introducing subsession identifiers ssid. For more clarity, throughout this article we use ssid for $(\mathsf{sid}, \mathsf{ssid})$. More explicitly, two ssid could theoretically be equal on different sessions sid, but by setting $\mathsf{ssid} := (\mathsf{sid}, \mathsf{ssid})$, we enforce the uniqueness of ssid. This uniqueness is necessary in the proofs provided in Appendices A and B.

The goal of the UC model is to emulate the protocol Π using the ideal functionality. If the emulation is performed such that the environment $\mathcal{Z}$ cannot distinguish (1) Π's outputs with possible interactions with an adversary $\mathcal{A}$ from (2) the outputs of dummy parties and a simulator interacting with the ideal functionality $\mathcal{F}$, then one can state that Π UC-emulates $\mathcal{F}$.

In our case, the protocols are Password-Based Authenticated Key Exchange (PAKE) and the ideal functionalities used throughout the paper specifically designed for PAKEs [CHK$^+$05,ACCP08] are $\mathcal{F}_{pwKE}$ and $\mathcal{F}_{pwKE\text{-}sA}$ (defined in Figures 3 and 4).

Ideal Functionality $\mathcal{F}_{pwKE}$**.** We present here the ideal functionality that is used for PAKEs. A detailed description of the functionality is provided in Figure 3. It consists of three types of queries: `NewSession`, `TestPwd` and `NewKey`:

- `NewSession` allows a party to initialize a connection to another opposing party using its password. The functionality $\mathcal{F}_{pwKE}$ uses this query to record the connection as well as the initial party's password.
- `TestPwd` models the unique online password test (online dictionary attacks) that is enabled through the execution of a PAKE. This query additionally impacts the view of the ideal functionality. Querying `TestPwd` changes the view of $\mathcal{F}_{pwKE}$ in the exchange of two parties by altering the behavior of the next query `NewKey`, according to the correct or incorrect guess.
- `NewKey` interface allows to give parties a session key consistent with the state of their record. If two fresh entities do not share/use the same password then

PAKE Ideal Functionality: $\mathcal{F}_{pwKE}$

Session Initialization

On $(\texttt{NewSession}, \mathsf{ssid}, \mathsf{pw}, P_i, P_j)$ from P_i:

- Sends $(\texttt{NewSession}, \mathsf{ssid}, P_i, P_j)$ to $\mathcal{A}$.
- If this is the first NewSession query, or if it is the second NewSession query and there is a record $(\mathsf{ssid}, P_j, P_i, \mathsf{pw}', \star) \in L$, then record $(\mathsf{ssid}, P_i, P_j, \mathsf{pw}, \mathsf{fresh})$ in L.

Active attack

Upon receiving a query $(\texttt{TestPwd}, \mathsf{ssid}, P_i, \mathsf{pw}')$ from the adversary $\mathcal{A}$, if there exists record $(\mathsf{ssid}, P_i, P_j, \mathsf{pw}, \mathsf{fresh}) \in L$, do:

- If $\mathsf{pw} = \mathsf{pw}'$ mark the record as compromised and reply to $\mathcal{A}$ with "*correct guess*".
- If $\mathsf{pw} \neq \mathsf{pw}'$, mark the record as interrupted and reply to $\mathcal{A}$ with "*wrong guess*".

Key Generation

Upon receiving a query $(\texttt{NewKey}, \mathsf{ssid}, P_i, \mathtt{SK})$ from $\mathcal{S}$, where $\mathtt{SK} \in \{\mathsf{keys}\}$, if there is a record of the form $(\mathsf{ssid}, P_i, P_j, \mathsf{pw}, \mathsf{status}) \in L$, for any value status, and this is the first NewKey query for P_i:

- If $\mathsf{status} = \mathsf{compromised}$, or if one of the players P_i or P_j is corrupted then send $(\mathsf{ssid}, \mathtt{SK})$ to P_i;
- If $\mathsf{status} = \mathsf{fresh}$ and there is a record $(\mathsf{ssid}, P_j, P_i, \mathsf{pw}', \mathsf{status}')$ with $\mathsf{pw}' = \mathsf{pw}$, and a session key $\mathtt{SK}'$ has been sent to P_j, that was fresh at that time, then send $(\mathsf{ssid}, \mathtt{SK}')$ to P_i;
- Else, choose a random key $\mathtt{SK}'$ whose length is k and send $(\mathsf{ssid}, \mathtt{SK}')$ to P_i.

Update the record as completed.

Fig. 3. $\mathcal{F}_{pwKE}$: the ideal Functionality of a PAKE.

$\mathcal{F}_{pwKE}$ does not give them the same key. Contrarily, if they do, this oracle returns the same key for both parties. However the behavior is more refined than that and takes account possible alterations from TestPwd (more details in Figure 5).

Ideal Functionality with server authentication $\mathcal{F}_{pwKE\text{-}sA}$**.** We present a variation $\mathcal{F}_{pwKE\text{-}sA}$ of the previous ideal functionality to add explicit server authentication. A detailed description of the functionality is provided in Figure 4.

- in each record in L (defined in Figure 3 and 4), we add a component $\mathsf{role} \in \{\mathsf{client}, \mathsf{server}\}$. From this point forward, L has components of the form $(\mathsf{ssid}, P_i, P_j, \mathsf{pw}, \mathsf{status}, \mathsf{role})$.
- If the client queries NewKey at a time when the server still has not queried NewKey in the same session ssid, then $\mathcal{F}_{pwKE\text{-}sA}$ does nothing.
- If P_i and P_j do not share the same password: then the client gets abort whatever the status is.

PAKE Ideal Functionality with server Authentication: $\mathcal{F}_{pwKE\text{-}sA}$

Session Initialization

On $(\texttt{NewSession}, \mathsf{ssid}, \mathsf{role}, \mathsf{pw}, P_i, P_j)$ from P_i:

- Sends $(\texttt{NewSession}, \mathsf{ssid}, \mathsf{role}, P_i, P_j)$ to $\mathcal{A}$.
- If this is the first $\texttt{NewSession}$ query, or if it is the second $\texttt{NewSession}$ query and there is a record $(\mathsf{ssid}, P_j, P_i, \mathsf{pw}', \star, \star) \in L$, then record $(\mathsf{ssid}, P_i, P_j, \mathsf{pw}, \mathsf{fresh}, \mathsf{role})$ in L.

Active attack

Upon receiving a query $(\texttt{TestPwd}, \mathsf{ssid}, P_i, \mathsf{pw}')$ from the adversary $\mathcal{A}$, if there exists record $(\mathsf{ssid}, P_i, P_j, \mathsf{pw}, \mathsf{fresh}, \mathsf{role}) \in L$, do:

- If $\mathsf{pw} = \mathsf{pw}'$ mark the record as $\mathsf{compromised}$ and reply to $\mathcal{A}$ with "*correct guess*".
- If $\mathsf{pw} \neq \mathsf{pw}'$, mark the record as $\mathsf{interrupted}$ and reply to $\mathcal{A}$ with "*wrong guess*".

Key Generation

Upon receiving a query $(\texttt{NewKey}, \mathsf{ssid}, P_i, \texttt{SK})$ from $\mathcal{S}$, where $\texttt{SK} \in \{\mathsf{keys}\}$, if there is a record of the form $(\mathsf{ssid}, P_i, P_j, \mathsf{pw}, \mathsf{status}, \mathsf{role}) \in L$, for any value status, and this is the first $\texttt{NewKey}$ query for P_i:

- if $\mathsf{role} = \mathsf{client}$
  - If $\mathsf{status} = \mathsf{compromised}$, or if one of the players P_i or P_j is corrupted and there exists two records $(\mathsf{ssid}, P_i, P_j, \mathsf{pw}, \mathsf{client})$ and $(\mathsf{ssid}, P_j, P_i, \mathsf{pw}, \mathsf{server})$ then send $(\mathsf{ssid}, \texttt{SK})$ to P_i;
  - Else if $\mathsf{status} = \mathsf{fresh}$ and there is a record $(\mathsf{ssid}, P_j, P_i, \mathsf{pw}', \mathsf{client}')$ with $\mathsf{pw}' = \mathsf{pw}$, and a session key $\texttt{SK}'$ has been sent to P_j, that was fresh at that time, then send $(\mathsf{ssid}, \texttt{SK}')$ to P_i. Else if $\mathsf{pw}' \neq \mathsf{pw}$, choose a random key $\texttt{SK}'$ whose length is k and send $(\mathsf{ssid}, \texttt{SK}')$ to P_i.
  - Else if $\mathsf{status} = \mathsf{fresh}$, and if no record $\mathsf{completed}$ record exists for P_j in ssid do nothing.
  - Else if $\mathsf{status} = \mathsf{interrupted}$, send $(\mathsf{ssid}, \mathsf{error})$ to P_i.
- if $\mathsf{role} = \mathsf{server}$
  - If $\mathsf{status} = \mathsf{compromised}$, or if one of the players P_i or P_j is corrupted then send $(\mathsf{ssid}, \texttt{SK})$ to P_i;
  - Else if $\mathsf{status} = \mathsf{fresh}$ or $\mathsf{status} = \mathsf{interrupted}$, choose a random key $\texttt{SK}' \in \{keys\}$ and send $(\mathsf{ssid}, \texttt{SK}')$ to P_i.

Update the record as $\mathsf{completed}$.

Fig. 4. $\mathcal{F}_{pwKE\text{-}sA}$: the ideal Functionality of a PAKE with server explicit authentication.

Model. To prove that a protocol UC-emulates $\mathcal{F}_{pwKE}$ or $\mathcal{F}_{pwKE\text{-}sA}$, we first need to set the model and assumptions for the proof. In this paper, we consider the Random Oracle Model (ROM) and the Ideal Cipher (IC) model. We also make use of the erasure model and assume that the adversary $\mathcal{A}$ is able to perform either adaptive or static corruptions, depending on the protocol.

Random Oracle. We use the definition of ROM introduced by Hofheinz and Müller-Quade [HM04] recalled in Figure 12 from Appendix C. This assumption provides a powerful tool that coherently responds to queries and generates answers that are uniformly random and independent of the input of the query.

Ideal Cipher. The ideal cipher model was first introduced in [BPR00]. It considers that a cipher behaves as a perfectly independent random permutation for every key used.We will generalize it a little bit by differentiating the input set and the output set, and then considering random bijections for every key. This model is presented in more details in Figure 13 from Appendix C.

Corruption. As mentioned earlier, we consider two types of corruptions in this paper: static corruptions and adaptive corruptions. Static corruptions allow the adversary to obtain the password of a party prior to the execution of the protocol. This means that during a simulation, the simulator knows which parties have been corrupted. Adaptive corruptions allow the adversary to corrupt any party during the execution of the protocol by revealing the password and internal state of the party. The adaptive corruption functionality is defined in Figure 14 of Appendix C.

Erasure model. The erasure model is a simple but powerful assumption. In this model, we assume that any internal information that is no longer useful ceases to exist. Therefore, in the event of information leakage, or adaptive corruption, previous internal information is not leaked as it no longer exists.

4 Two Pieces of One Cake: Study of EKE and OEKE

In this study, we propose to examine the use of KEM with specific properties in the context of EKE and One-Way Encrypted Key Exchange (OEKE). We first introduce an evolved version of EKE called CAKE, that provides implicit authentication only, and then extend it to OEKE using a variant called OCAKE, that additionally provides explicit authentication of the receiver. We prove the security of these protocols in the Universal Composability (UC) model, assuming three properties of the KEM: semantic security, fuzziness, and anonymity. These two studies offer a balance between security properties and efficiency: CAKE handles adaptive corruptions, while OCAKE is proven secure in a relaxed model that only allows static corruptions to provide explicit authentication.

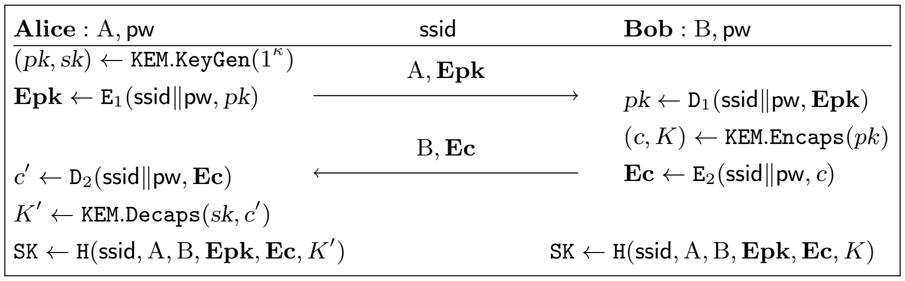
Alice : A, pw
ssid
Bob : B, pw
$(pk, sk) \leftarrow \mathtt{KEM.KeyGen}(1^\kappa)$
$\mathbf{Epk} \leftarrow \mathtt{E}_1(\mathsf{ssid}\|\mathsf{pw}, pk)$
A, **Epk**
$pk \leftarrow \mathtt{D}_1(\mathsf{ssid}\|\mathsf{pw}, \mathbf{Epk})$
$(c, K) \leftarrow \mathtt{KEM.Encaps}(pk)$
$\mathbf{Ec} \leftarrow \mathtt{E}_2(\mathsf{ssid}\|\mathsf{pw}, c)$
B, **Ec**
$c' \leftarrow \mathtt{D}_2(\mathsf{ssid}\|\mathsf{pw}, \mathbf{Ec})$
$K' \leftarrow \mathtt{KEM.Decaps}(sk, c')$
$\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, \mathrm{A}, \mathrm{B}, \mathbf{Epk}, \mathbf{Ec}, K')$
$\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, \mathrm{A}, \mathrm{B}, \mathbf{Epk}, \mathbf{Ec}, K)$

Fig. 5. CAKE with $(\mathtt{E}_1, \mathtt{D}_1)$, $(\mathtt{E}_2, \mathtt{D}_2)$ two pairs of ideal ciphers. $\mathtt{E}_1$ is a bijection from $\mathcal{P}$ to $\mathcal{P}'$ while $\mathtt{E}_2$ is a bijection from $\mathcal{C}$ to $\mathcal{C}'$.

4.1 CAKE

In this subsection, we present a study of the K(EM)-EKE protocol, referred to as CAKE. This protocol is based on the use of generic KEM in EKE and is the most conservative of the two constructions we propose.

To study CAKE properly, as well as expressing the necessary properties on the underlying KEM, we first fix a KEM (KeyGen, Encaps, Decaps) with the sets $\mathcal{SK}$, $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{K}$ as in Definition 1. Next, we define two ideal cipher pairs $(\mathtt{E}_1, \mathtt{D}_1)$ and $(\mathtt{E}_2, \mathtt{D}_2)$. We additionally define a set of keys Key and two sets $\mathcal{P}'$ and $\mathcal{C}'$ respectively bijections from $\mathcal{P}$ and $\mathcal{C}$ where both of them offer easy uniform sampling:

$$\begin{array}{ll} \mathtt{E}_1 : \mathtt{Key} \times \mathcal{P} \to \mathcal{P}' & \qquad \mathtt{E}_2 : \mathtt{Key} \times \mathcal{C} \to \mathcal{C}' \\ \mathtt{D}_1 : \mathtt{Key} \times \mathcal{P}' \to \mathcal{P} & \qquad \mathtt{D}_2 : \mathtt{Key} \times \mathcal{C}' \to \mathcal{C} \end{array}$$

The actual keys of the ideal ciphers are the concatenations of the ssid and the passwords, to ensure independent bijections between different executions of the protocol. We introduce a description of CAKE in Figure 5 along with its security theorem based on the fuzziness and anonymity of the underlying KEM in the ROM and IC models, while allowing adaptive corruptions.

Theorem 1. *Let* $(\mathtt{E}_1, \mathtt{D}_1)$, $(\mathtt{E}_2, \mathtt{D}_2)$ *be two pairs of ideal ciphers and* $\mathtt{H}$ *be a random oracle. We note* $q_{\mathtt{D}_1}$ *(resp.* $q_{\mathtt{D}_2}$*) the maximal number of queries to the decryption oracle* $\mathtt{D}_1$ *(resp.* $\mathtt{D}_2$*). We also note* $q_{\mathtt{E}_1}$ *(resp.* $q_{\mathtt{E}_2}$*) the maximal number of queries to the encryption oracle* $\mathtt{E}_1$ *(resp.* $\mathtt{E}_2$*) explicitly asked by the adversary. Finally, we note* q_s *the number of sessions. The* CAKE *protocol described in Figure 5 using* KEM*, a key encapsulation mechanism that is both fuzzy (Def. 3) and anonymous (Def. 4) ensuring semantic security, UC-emulates* $\mathcal{F}_{pwKE}$ *in the erasure model with adaptive corruptions.*
More precisely, if we define $\mathsf{Adv}^{\mathrm{cake}}_{\mathtt{KEM}}(\mathcal{A})$ *the advantage of an adversary* $\mathcal{A}$ *to break*

the above claim, it is bounded by

$$\begin{aligned}&(2q_s + q_{\mathtt{D}_1} + q_{\mathtt{D}_2}) \cdot \mathsf{Adv}^{\text{ind}}_{\mathtt{KEM}}(t)\\ &+ (q_s + q_{\mathtt{D}_1}) \cdot \mathsf{Adv}^{\text{fuzzy}}_{\mathtt{KEM}}(t) + q_{\mathtt{D}_1} \cdot (q_s + q_{\mathtt{D}_2}) \cdot \mathsf{Adv}^{\text{ano}}_{\mathtt{KEM}}(t)\\ &+ q_{\mathtt{H}} \cdot q_s \cdot 2^{-\lambda_k} + q_{\mathtt{E}_1}^2 \cdot 2^{-\lambda_p - 1} + q_{\mathtt{E}_2}^2 \cdot 2^{-\lambda_c - 1},\end{aligned}$$

where λ_k is the bit-length of the encapsulated keys, λ_p the bit-length of the public keys, and λ_c the bit-length of the ciphertexts, for the KEM *scheme.*

Sketch of Proof: In the subsequent games, we denote $\Pr[\mathbf{G}]$ the probability for the environment $\mathcal{Z}$ to output 1 in the simulated game $\mathbf{G}$. The goal is to prove that $\Pr[\mathbf{G}]$ is close to the probability to output 1 in the ideal game, while starting from the real game $\mathbf{G}_0$. The sequence of games will end with $\mathbf{G}_9$ that only uses the ideal functionality $\mathcal{F}_{pwKE}$, and is thus the ideal game. The complete proof can be found in the Appendix A. We present here a sketch of proof:

$\mathbf{G}_0$: $\mathcal{R}$eal world protocol using the following assumptions: erasure model, random oracle, ideal cipher, adaptive corruption and lastly a fuzzy and anonymous KEM, which is also indistinguishable.

$\mathbf{G}_1$: Honest simulation of the random oracle H and the pairs of ideal ciphers $(\mathtt{E}_1, \mathtt{D}_1)$ and $(\mathtt{E}_2, \mathtt{D}_2)$ from Figure 5, where we abort in case of collision during explicit encryption calls. Additionally, a private simulation of a random oracle $\mathtt{H}^*$ used for the simulation when $\mathcal{S}$ cannot extract private information.

$\mathbf{G}_2$: Embedding of the secrets during the simulation of $\mathtt{D}_1$ and $\mathtt{D}_2$.

$\mathbf{G}_3$: Simulation of Alice's initialization with $\mathtt{D}_1$ instead of $\mathtt{E}_1$.

$\mathbf{G}_4$: Simulation of Bob's answer with $\mathtt{D}_2$ instead of $\mathtt{E}_2$.

$\mathbf{G}_5$: Preparation of Alice's reaction, by anticipating all the possible public keys decrypted by $\mathtt{D}_1$ when a query is asked to $\mathtt{D}_2$.

$\mathbf{G}_6$: Simulation of Alice's reaction, using the previous simulation of $\mathtt{D}_2$.

$\mathbf{G}_7$: Random session keys, where we replace all the unknown keys SK by random values.

$\mathbf{G}_8$: Adaptive corruptions, where we program the random oracle H and provide the secret values in case of corruption.

$\mathbf{G}_9$: Using on queries from $\mathcal{F}_{pwKE}$ to detail the simulator in the ideal world.

A precise simulator is defined in Appendix A with Figures 7, 8 and 9. □

Remark 1. Instantiated with the KEM derived from ElGamal presented in Section 2, CAKE-ElGamal is exactly the famous EKE [BM92]. But the proof technique actually differs because ElGamal enjoys perfect fuzziness (or smoothness) and perfect anonymity, which facilitate the EKE security proof. However, post-quantum algorithms cannot validate all these strong properties.

4.2 OCAKE

In this subsection, we modify the above CAKE protocol by adding explicit authentication of the receiver, which allows to remove one encryption. The modifications are based on the OEKE [BCP03] protocol but with generic KEM protocols

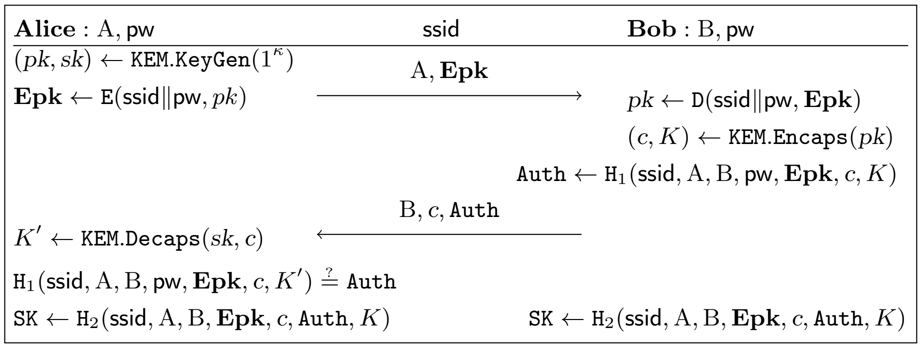
Alice : A, pw
ssid
Bob : B, pw
(pk, sk) ← KEM.KeyGen(1^κ)
Epk ← E(ssid‖pw, pk)
A, Epk
pk ← D(ssid‖pw, Epk)
(c, K) ← KEM.Encaps(pk)
Auth ← H1(ssid, A, B, pw, Epk, c, K)
B, c, Auth
K′ ← KEM.Decaps(sk, c)
H1(ssid, A, B, pw, Epk, c, K′) ?= Auth
SK ← H2(ssid, A, B, Epk, c, Auth, K)
SK ← H2(ssid, A, B, Epk, c, Auth, K)

Fig. 6. OCAKE with $(\mathtt{E}, \mathtt{D})$ an ideal cipher. $\mathtt{E}$ is a bijection from $\mathcal{P}$ to $\mathcal{P}'$.

instead of a Diffie-Hellman based key exchange.

In this setting, we only handle static corruptions in the security model. Indeed, it would have been possible to include adaptive corruptions in the security model with a statistical notion for the anonymity, i.e. perfect anonymity. But the computational property of our anonymity definition (see Definition 4) is more realistic for post-quantum KEM protocols. And thus here, adaptive corruptions cannot be handled by our proof in the UC-framework: in case of honest transcripts, we must generate a random c, on behalf of Bob. In case of Alice's corruption, one can program $\mathtt{E}$ in order to set a specific (pk, sk), but if the simulator commits on a specific c, then it cannot remain consistent. In particular, the adversary could have tried many passwords when decrypting **Epk**, hence one cannot anticipate the public key for c.

In order to thoroughly study this approach, we outline the modifications in Fig. 6. We remove one encryption on the server flow to change it into an authentication. However for the sake of the proof, we have to slightly change how the hash query is usually done. Instead of using the public transcript, we use part of the secret information for the sake of the simulation in the security proof. Lastly we use $\mathcal{F}_{pwKE\text{-}sA}$ in Fig. 4: the PAKE ideal functionality with explicit authentication of the server.

Theorem 2. *Let* $(\mathtt{E}, \mathtt{D})$ *be a pair of ideal cipher. Let* $\mathtt{H}_1$ *and* $\mathtt{H}_2$ *be two random oracles. We note* $q_{\mathtt{D}}$ *the maximal number of queries to the decryption oracle* $\mathtt{D}$*. We also note* $q_{\mathtt{E}}$ *the maximal number of queries to the encryption oracle* $\mathtt{E}$*, explicitly asked by the adversary. And we note* q_s *the number of sessions. The OCAKE protocol described in Figure 6 using* KEM*, a key encapsulation mechanism that is both fuzzy (Def. 3) and anonymous (Def. 4) while ensuring semantic security, UC-emulates* $\mathcal{F}_{pwKE\text{-}sA}$ *in the erasure model with static corruptions.*
More precisely, if we define $\mathsf{Adv}^{\mathrm{ocake}}_{\mathtt{KEM}}(\mathcal{A})$ *the advantage of an adversary* $\mathcal{A}$ *to*

break the above claim is bounded by

$$(q_s + q_{\mathtt{D}}) \cdot \mathsf{Adv}^{\text{fuzzy}}_{\mathtt{KEM}}(t) + (q_s + q_{\mathtt{D}} + 1) \cdot \mathsf{Adv}^{\text{ind}}_{\mathtt{KEM}}(t) + q_{\mathtt{D}} \cdot \mathsf{Adv}^{\text{ano}}_{\mathtt{KEM}}(\mathcal{A})$$
$$+ (q_{\mathtt{H}_1} + 2q_s)^2 \cdot 2^{-\lambda_{\mathtt{H}_1}-1} + q_{\mathtt{E}}^2 \cdot 2^{-\lambda_p-1} + (q_{\mathtt{H}_1} + q_{\mathtt{H}_2}) \cdot q_s \cdot 2^{-\lambda_k},$$

where λ_k is the bit-length of the encapsulated keys, λ_p the bit-length of the public keys for the KEM *scheme, and $\lambda_{\mathtt{H}_1}$ the bit-length of the authentication tag.*

Sketch of Proof: Similarly to the proof of Theorem 1, in the subsequent games, we denote $\Pr[\mathbf{G}]$ the probability for the environment $\mathcal{Z}$ to output 1 in the simulated game $\mathbf{G}$. The goal is to prove that $\Pr[\mathbf{G}]$ is close to the probability to output 1 in the ideal game, while starting from the real game $\mathbf{G}_0$. The sequence of games will end with $\mathbf{G}_8$ that only uses the ideal functionality $\mathcal{F}_{pwKE\text{-}sA}$, and is thus the ideal game. The complete proof can be found in the Appendix B. We present here a sketch of proof:

- $\mathbf{G}_0$**:** $\mathcal{R}eal$ world protocol using the following assumptions: erasure model, random oracle, ideal cipher, static corruption and lastly a fuzzy and anonymous KEM, which is also indistinguishable.
- $\mathbf{G}_1$**:** Honest simulation of two random oracles $\mathtt{H}_1$, $\mathtt{H}_2$ and an ideal cipher $(\mathtt{E}, \mathtt{D})$ from Figure 6, where we abort in case of collision during explicit encryption calls. Additionally a private simulation of each random oracle $\mathtt{H}_1^*$, $\mathtt{H}_2^*$ used for the simulation when $\mathcal{S}$ does not know any passwords. We also exclude collisions on $\mathtt{H}_1$ and $\mathtt{H}_1^*$.
- $\mathbf{G}_2$**:** Embedding of the secret keys during the simulation of D.
- $\mathbf{G}_3$**:** Simulation of an adversary finding Auth by chance.
- $\mathbf{G}_4$**:** Simulation of Alice's initialization with D instead of E.
- $\mathbf{G}_5$**:** Simulation of Bob's answer with $(c, \mathtt{Auth})$.
- $\mathbf{G}_6$**:** Simulation of Alice's reaction, using the Auth and the abortion in case the authentication is not verified.
- $\mathbf{G}_7$**:** Random session keys, where we replace all the unknown authentication tags Auth and keys SK by random values, except for correctly guessed passwords
- $\mathbf{G}_8$**:** Using on queries from $\mathcal{F}_{pwKE\text{-}sA}$ to detail the simulator in the ideal world.

A precise simulator is defined in Appendix B with Figures 10 and 11. □

Remark 2. Comparatively to remark 1, instantiated with the KEM derived from ElGamal presented in Section 2, OCAKE-ElGamal is exactly OEKE [ACCP08].

Additional remarks :

Removing the cipher on *pk* and keeping it on *c* like OEKE is not possible. To follow strictly its framework one would need a perfectly anonymous KEM otherwise the client could construct a subset attack using the gap of a misconstructed cipher on decryption. Furthermore, we place our study in the context of quantum-secure KEM and none of them allows for perfect anonymity. Therefore the strict OEKE framework is not an enticing approach for long-term usability.

- This protocol is only secure against static corruptions. An adversary allowed to apply adaptive corruptions could corrupt the client a reception of $(c, \texttt{Auth})$ before it computes `Decaps`. The adversary would then obtain *sk* because `Decaps` needs it and implies that the erasure is not applied. Knowing *sk*, a random $c \leftarrow\!\!\$\ \mathcal{C}$ is easily recognizable from a honestly built one leading the adversary to distinguish the simulation in the above proof.
- Adding an authentication of the client afterwards for mutual authentication is entirely possible. The proof extensively use tricks before the derivation of the session key to either extract private information or to send indistinguishable random elements. Since this is done before, either the client would send a honest authentication, a perfectly indistinguishable one or a recognizable wrong one.

5 Crystal-Kyber

5.1 Security Properties

Crystals-`Kyber` has been introduced in Section 2. For the following results, we set $\mathcal{P} = \{0,1\}^\kappa \times \mathcal{R}_q^k$, $\mathcal{SK} = \beta_\eta^k$, $\mathcal{C} = \mathcal{R}_q^k \times \mathcal{R}_q$ and $\mathcal{K} = \{0,1\}^n$. First of all, we recall the indistinguishability property of Crystals-`Kyber` [BDK+18]:

Lemma 1. `Kyber` *on parameters* (k, η, q, n) *is an indistinguishable* `KEM`*:*

$$\mathsf{Adv}^{\mathrm{ind}}_{\mathtt{Kyber}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{d-mlwe}}_{k,k,\eta}(t) + \mathsf{Adv}^{\mathsf{d-mlwe}}_{k+1,k,\eta}(t)$$

Next, let us verify the anonymity and fuzziness properties guaranteed by `Kyber`.

Lemma 2. *Crystals-*`Kyber` *is an anonymous* `KEM` *in* $\mathcal{C} = \mathcal{R}_q^k \times \mathcal{R}_q$*:*

$$\mathsf{Adv}^{\mathrm{ano}}_{\mathtt{Kyber}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{d-mlwe}}_{k,k,\eta}(t) + \mathsf{Adv}^{\mathsf{d-mlwe}}_{k+1,k,\eta}(t)$$

Proof. Let sample a public key $pk \leftarrow\!\!\$\ (\mathbf{A}, \mathbf{b})$, by definition of `Kyber` in Figure 1

$$c = (\mathbf{u}, v) \text{ with } \begin{cases} \mathbf{u} = \mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}' \\ v = \mathbf{b}^T \cdot \mathbf{r} + e'' + \left\lceil \frac{q}{2} \right\rfloor \cdot m \end{cases}$$

We can rewrite c as:

$$\begin{bmatrix} \mathbf{u} \\ v \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{A} \\ \mathbf{b} \end{bmatrix}^T \mathbf{r} + \begin{bmatrix} \mathbf{e}' \\ e'' + \left\lceil \frac{q}{2} \right\rfloor \cdot m \end{bmatrix}$$

It forms a Module-LWE instance $\left(\begin{bmatrix} \mathbf{A} \\ \mathbf{b} \end{bmatrix}^T, \begin{bmatrix} \mathbf{u} \\ v \end{bmatrix} \right)$ provided that $(\mathbf{A}, \mathbf{b})$ is uniformly random on $\mathcal{R}_q^{k\times k} \times \mathcal{R}_q^k$ which is true under the $\mathbf{d\text{-}MLWE}_{k,k,\eta}$ assumption. This directly gives:

$$\mathsf{Adv}^{\mathrm{ano}}_{\mathtt{Kyber}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{d-mlwe}}_{k+1,k,\eta}(t) + \mathsf{Adv}^{\mathsf{d-mlwe}}_{k,k,\eta}(t)$$

□

To prove anonimity in lemma 2, `Kyber` needs to ensure the decisional MLWE argument, therefore:

Corollary 1. *Crystals-Kyber is a fuzzy* KEM *with* $\mathcal{P} = \mathcal{R}_q^k$*:*

$$\mathsf{Adv}^{\text{fuzzy}}_{\texttt{Kyber}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{d-mlwe}}_{k,k,\eta}(t)$$

Theorem 3. *Let* $(\mathtt{E}_1, \mathtt{D}_1)$, $(\mathtt{E}_2, \mathtt{D}_2)$ *be two pairs of ideal ciphers, and* $\mathtt{H}$ *a random oracle. We note* $q_{\mathtt{D}_1}$ *(resp.* $q_{\mathtt{D}_2}$*) the maximal number of queries to the decryption oracle* $\mathtt{D}_1$ *(resp.* $\mathtt{D}_2$*), explicitly asked by the adversary, and* q_s *the number of session. The* CAKE *protocol from Figure 5 instantiated with* `Kyber` *UC-emulates* $\mathcal{F}_{pwKE}$ *in the erasure model with adaptive corruptions:*

$$\begin{aligned}\mathsf{Adv}^{\text{cake}}_{\texttt{Kyber}}(\mathcal{A}) \leq{}& ((5q_s + 3q_{\mathtt{D}_1} + 2q_{\mathtt{D}_2}) + 2q_{\mathtt{D}_1} \cdot (q_s + q_{\mathtt{D}_2})) \cdot \mathsf{Adv}^{\mathsf{d-mlwe}}_{k+1,k,\eta}(t)\\ &+ q_{\mathtt{H}} \cdot q_s \cdot 2^{-n} + q^{-kn} \cdot (q^2_{\mathtt{E}_1} \cdot 2^{-\kappa} + q^2_{\mathtt{E}_2} \cdot q^{-n})/2\end{aligned}$$

Theorem 4. *Let* $(\mathtt{E}, \mathtt{D})$ *be an ideal cipher, and* $\mathtt{H}_1$, $\mathtt{H}_2$ *two random oracles. We note* $q_{\mathtt{D}}$ *the maximal number of queries to the decryption oracle* $\mathtt{D}$*, explicitly asked by the adversary, and* q_s *the number of session. The* OCAKE *protocol from Figure 6 instantiated with* `Kyber` *UC-emulates* $\mathcal{F}_{pwKE\text{-}sA}$ *in the erasure model with static corruptions:*

$$\begin{aligned}\mathsf{Adv}^{\text{ocake}}_{\texttt{Kyber}}(\mathcal{A}) \leq{}& 2 \cdot q_{\mathtt{D}} \cdot (\mathsf{Adv}^{\mathsf{d-mlwe}}_{k+1,k,\eta}(t)) + 3 \cdot q_{\mathtt{D}} \cdot (\mathsf{Adv}^{\mathsf{d-mlwe}}_{k,k,\eta}(t))\\ &+ (q_{\mathtt{H}_1} + q_{\mathtt{H}_2}) \cdot q_s \cdot 2^{-n} + q^2_{\mathtt{E}} \cdot 2^{-\kappa} + q_{\mathtt{H}_1} \cdot 2^{-n}\end{aligned}$$

5.2 Instantiation of the Block Cipher

To prove the UC-security of both CAKE-Kyber and OCAKE-Kyber, the ideal cipher model is crucial. More precisely it needs to ensure that finding a collision on the encryption is statistically impossible without querying a decryption oracle. It removes both the following approaches out of the equation: stream cipher and one time pad. The conception of a relevant block cipher for our transformations is actually nontrivial. In fact, the underlying sets, like $\mathcal{R}_q$, are not convenient for building a symmetric block cipher statistically following the necessary ideal properties. We present here a solution issued from known ad-hoc techniques. We believe that it can be improved for better performance but this task is left as future work.

To keep light notations, we do not encrypt the seed of $\mathbf{A}$, and thus consider the encryption of an element in $\mathcal{R}_q^k \sim \mathbb{Z}_q^{n\times k}$ for the public key. We can do the same with $\mathcal{R}_q^k \times \mathcal{R}_q \sim \mathbb{Z}_q^{n\times(k+1)}$ for the ciphertext.

To encrypt $pk \in \mathcal{R}_q^k \sim \mathbb{Z}_q^{n\times k}$, we can first encode pk into $\{0, \dots q^{nk} - 1\}$, and then use a block cipher on ℓ-bits, such that $2^{\ell-1} \leq q^{nk} < 2^\ell$. We thus have an encoding/decoding from $\mathcal{R}_q^k$ to $\{0,1\}^\ell$ that can be seen as a superset of $\{0, \dots q^{nk} - 1\}$. Let us thus consider all these encodings equivalent.

From $(\mathtt{E}, \mathtt{D})$ on ℓ-bit blocks and κ-bit keys, we can build a permutation onto the restricted set $\{0, \dots q^{nk} - 1\}$: one defines the encryption scheme with key K on $\mathbf{b} \in \mathcal{R}_q^k \sim \{0, \dots q^{nk} - 1\}$ as $\mathtt{E}'_K(\mathbf{b}) = \mathtt{E}_K(\dots \mathtt{E}_K(\mathbf{b}) \dots)$, stopping at the first element in $\{0, \dots q^{nk} - 1\}$. Decryption works the same way, and will stop at the right place as all ignored intermediate values are outside the expected set.

Actually, the number of iterations will be small, as there is a probability less than $1/2$ at each step. This technique is vulnerable to timing attacks, but one can always include virtual loops.

We emphasize that this study is done without using the optimized `Kyber` (without) the `compression`, `decompression` functions. However, these two functions map elements of $\mathbb{Z}_q$ to $\mathbb{Z}_{q'}$ with $q' < q$, therefore the study remains similar.

5.3 Parameters

The bounds in Theorems 3 and 4 are slightly looser than the ones that constrain the choice of parameters for `Kyber`. Thus, some adaptations of the obtained security levels are necessary. We propose to recompute the security level for the set of parameters taken from `Kyber`'s last submission to the NIST [SAB+22] and choosing parameters that allow to reach around 100 bits of security against quantum adversaries. While this can be argued to be weak, PAKE are not used in highly critical applications but in highly efficient ones. Hence, 100 bits of security against quantum adversaries would constitute a mid to long-term security target.

We present in Table 1 the security estimations for CAKE-Kyber and OCAKE-Kyber obtained with the pq-crystal estimate [DS21] against a quantum adversary with `KYBER768` and `KYBER1024` parameters.

`Kyber` parameters	Bit-sec against quantum adversaries obtained with [DS21]
`Kyber1024`	**102**

CAKE − Kyber

`Kyber` parameters	Bit-sec against quantum adversaries obtained with [DS21]
`Kyber768`	**98**
`Kyber1024`	**162**

OCAKE − Kyber

Table 1. Bit security estimates of CAKE-Kyber and OCAKE-Kyber using parameters from version 3.0 of the NIST [SAB+22] against a quantum adversary. Estimation done using python script from pqcrystals github [DS21].

The security/efficiency trade-off between CAKE and OCAKE is confirmed in this example. According to Table 1, CAKE provides more conservative assumptions as an adaptive adversary are included in the model however it is less efficient that its OCAKE alternative.

6 Conclusion and perspectives

In this article we characterize the necessary properties for a key encapsulation mechanism to be used in a password authenticated key exchange and more precisely in both EKE and OEKE. Additionally we prove that these properties are respected by the newly standardized Kyber. To supplement this study we introduce a set of possible parameters for Kyber, ensuring around 100 bit of security. Lastly we propose a cipher respecting statistically ideal cipher properties for the application of both CAKE (Fig. 5) and (OCAKE Fig. 6).
While our work focuses on post-quantum alternatives, one could improve our results by supposing a random self-reducible KEM (like El-Gamal). Random self-reducibility implies that arbitrarily many independant instances can be reduced to only one such instance. Although current post-quantum schemes are not self-reducible KEM but such an assumption would lead to tighter reductions and it would allow for SPEKE or CPace constructions to be generalized to more KEM protocols.

Acknowledgements. We would like to thank Olivier Blazy and Henri Gilbert respectively for useful discussions about password-authenticated key exchange protocols and their security and symmetric encryption for the PAKE practicability. This work was supported in part by the French Programme d'Investissement d'Avenir (PIA) under national project RESQUE. The first author was also supported by ANRT under the program CIFRE N° 2021/0645.

References

ABC$^+$22. Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions.

ACCP08. Michel Abdalla, Dario Catalano, Céline Chevalier, and David Pointcheval. Efficient two-party password-based key exchange protocols in the UC framework. In Tal Malkin, editor, *CT-RSA 2008*, volume 4964 of *LNCS*, pages 335–351. Springer, Heidelberg, April 2008.

ADPS16. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. NewHope without reconciliation. Cryptology ePrint Archive, Report 2016/1157, 2016. https://eprint.iacr.org/2016/1157.

AHH21. Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of CPace. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 711–741. Springer, Heidelberg, December 2021.

AHH22. Michel Abdalla, Björn Haase, and Julia Hesse. CPace, a balanced composable PAKE. Internet-Draft draft-irtf-cfrg-cpace-06, Internet Engineering Task Force, July 2022. Work in Progress.

BCP03. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Security proofs for an efficient password-based key exchange. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM CCS 2003*, pages 241–250. ACM Press, October 2003.

BCV19. Olivier Blazy, Céline Chevalier, and Quoc Huy Vu. Post-quantum uc-secure oblivious transfer in the standard model with adaptive corruptions. In *Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES 2019, Canterbury, UK, August 26-29, 2019*, pages 28:1–28:6. ACM, 2019.

BDK$^+$18. Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - kyber: A cca-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 353–367. IEEE, 2018.

BM92. Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.

BPR00. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000.

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

CHK$^+$05. Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In

Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Heidelberg, May 2005.

CR03. Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, Heidelberg, August 2003.

DHP$^+$18. Pierre-Alain Dupont, Julia Hesse, David Pointcheval, Leonid Reyzin, and Sophia Yakoubov. Fuzzy password-authenticated key exchange. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 393–424. Springer, Heidelberg, April / May 2018.

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

DKRV18. Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 18*, volume 10831 of *LNCS*, pages 282–305. Springer, Heidelberg, May 2018.

DS21. Léo Ducas and John Schanck. pq-crystals/security-estimates. https://github.com/pq-crystals/security-estimates, 2021.

ElG85. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

GDLL17. Xinwei Gao, Jintai Ding, Jiqiang Liu, and Lin Li. Post-quantum secure remote password protocol from RLWE problem. Cryptology ePrint Archive, Report 2017/1196, 2017. https://eprint.iacr.org/2017/1196.

HM04. Dennis Hofheinz and Jörn Müller-Quade. Universally composable commitments using random oracles. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 58–76. Springer, Heidelberg, February 2004.

LS15. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 75(3):565–599, 2015.

Mac01. Philip MacKenzie. On the security of the SPEKE password-authenticated key exchange protocol. Cryptology ePrint Archive, Report 2001/057, 2001. https://eprint.iacr.org/2001/057.

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

PAA$^+$19. Thomas Poppelmann, Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Peter Schwabe, Douglas Stebila, Martin R. Albrecht, Emmanuela Orsini, Valery Osheter, Kenneth G. Paterson, Guy Peer, and Nigel P. Smart. NewHope. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

Reg06. Oded Regev. Lattice-based cryptography (invited talk). In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 131–141. Springer, Heidelberg, August 2006.

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

Sch17. Jörn-Marc Schmidt. Requirements for password-authenticated key agreement (PAKE) schemes. *RFC*, 8125:1–10, 2017.

ZY17. Jiang Zhang and Yu Yu. Two-round PAKE from approximate SPH and instantiations from lattices. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 37–67. Springer, Heidelberg, December 2017.

Supplementary Material

These appendices are for the reviewer convenience.

A Proof of Theorem 1

Game $\mathbf{G_0}$: In this game we present a formalization of the CAKE protocol using the random oracle model and the ideal cipher model. Our simulation is set in the erasure model, in which secret information is erased from the memory of each party when it is no longer needed during the protocol execution. The protocol is executed in an adversarial environment, denoted as $\mathcal{Z}$, in which the parties can be adaptively corrupted by an adversary $\mathcal{A}$ to leak their private state. In our simulation, Alice is referred to as P_i (the initiator) and Bob is referred to as P_j (the responder). Additionally, pw_A represents Alice's password and pw_B represents Bob's password, while pw represents an arbitrary password usually used by the adversary.

Game $\mathbf{G_1}$: Simulation of $\mathcal{F}_{IC}$ and $\mathcal{F}_{RO}$. This game builds the simulation from $\mathcal{S}$ of the different oracles namely: the ideal cipher and the random oracle. It is subdivisided into three subgames and each subgame represents respectively the simulation of the random oracle and two differently modeled ideal ciphers, starting from $\mathbf{G}_0$, with $\mathbf{G}_{0.1}$, $\mathbf{G}_{0.2}$, and $\mathbf{G}_{0.3} = \mathbf{G}_1$:

Game $\mathbf{G}_{0.1}$*:* In this subgame, $\mathcal{S}$ models the random oracle $\mathtt{H}$, where on each query the oracle returns a uniformly random answer. To remain consistent with previous answers $\mathcal{S}$ uses a list $\Lambda_{\mathtt{H}}$ of tuples $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K, \mathtt{SK})$. This list is initially set as empty and grows as the number of queries to $\mathtt{H}$ piles up in the different sessions. On query $\mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$, $\mathcal{S}$ uses $\Lambda_{\mathtt{H}}$ to simulate it as follows:

– If a record $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K, \mathtt{SK})$ exists in $\Lambda_{\mathtt{H}}$, $\mathcal{S}$ returns $\mathtt{SK}$.
– Else, $\mathcal{S}$ samples a random $\mathtt{SK}$, records $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K, \mathtt{SK})$ in $\Lambda_{\mathtt{H}}$, and returns $\mathtt{SK}$.

However throughout the simulation, $\mathcal{S}$ will have to simulate $\mathtt{H}$ while not knowing K, for generating $\mathtt{SK}$. $\mathcal{S}$ uses a private oracle $\mathtt{H}^*$ to record values in a list $\Lambda_{\mathtt{H}^*}$ of items $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathsf{status}, \mathtt{SK})$. $\mathcal{S}$ simulates $\mathtt{H}^*$ as follows on input $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathsf{status})$, where status can be $\mathsf{success}$, fail_A, or fail_B, where both $\mathsf{success}$ would lead to the same key $\mathtt{SK}$, whereas a failure will lead to independent keys:

– If a record $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathsf{status}, \mathtt{SK})$ exists in $\Lambda_{\mathtt{H}^*}$, $\mathcal{S}$ returns $\mathtt{SK}$.
– Else, $\mathcal{S}$ samples a random $\mathtt{SK}$, records $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathsf{status}, \mathtt{SK})$ in $\Lambda_{\mathtt{H}^*}$, and returns $\mathtt{SK}$.

This simulation is perfectly identical to $\mathbf{G}_0$.

Game $\mathbf{G}_{0.2}$*:* In this game, we simulate an ideal cipher from the first exchange of the protocol. The simulation of this oracle needs to be consistent, meaning that any query that has already been asked should return the same answer as the first time it was asked. Additionally, the simulation needs to capture the properties of an ideal cipher, which means simulating each encryption as a random bijection for each key (actually, for each $\mathsf{ssid}\|\mathsf{pw}$). But for the following simulation, we will need to avoid collisions during adversary's encryption with different inputs. Then, the simulator uses a list called Λ_1 for encryption and decryption queries on each oracle. Λ_1 is composed of tuples of the form $(\mathsf{ssid}, \mathsf{pw}, pk, sk, \mathtt{E}_1 \vee \mathtt{D}_1, \mathbf{Epk})$, even if the component sk will only appear later. $\mathcal{S}$ simulates $\mathtt{E}_1$ and $\mathtt{D}_1$ as follows:

- On $\mathtt{E}_1(\mathsf{ssid}\|\mathsf{pw}, pk)$:
  - If there exists a record $(\mathsf{ssid}, \mathsf{pw}, pk, \star, \star, \mathbf{Epk}) \in \Lambda_1$ then $\mathcal{S}$ returns $\mathbf{Epk}$.
  - Else, $\mathcal{S}$ samples $\mathbf{Epk} \leftarrow\!\!\$\ \mathcal{P}'$. If $\mathbf{Epk}$ already exists in Λ_1, $\mathcal{S}$ aborts, else $\mathcal{S}$ records $(\mathsf{ssid}, \mathsf{pw}, pk, \perp, \mathtt{E}_1, \mathbf{Epk})$ in Λ_1 and returns $\mathbf{Epk}$.
- On $\mathtt{D}_1(\mathsf{ssid}\|\mathsf{pw}, \mathbf{Epk})$:
  - If there is a record $(\mathsf{ssid}, \mathsf{pw}, pk, \star, \star, \mathbf{Epk}) \in \Lambda_1$, $\mathcal{S}$ returns pk.
  - Else, $\mathcal{S}$ samples $pk \leftarrow\!\!\$\ \mathcal{P}$, records $(\mathsf{ssid}, \mathsf{pw}, pk, \perp, \mathtt{D}_1, \mathbf{Epk})$ in Λ_1, and returns pk.

Analysis: Under the assumption of the Ideal Cipher model depicted by its ideal functionality, $\mathcal{Z}$ can distinguish the real execution of the protocol from this game if $\mathcal{S}$ aborts. Assuming the cardinal of $\mathcal{P}$ is 2^{λ_p}, and if $\mathcal{A}$ makes up to $q_{\mathtt{E}_1}$ queries to the encryption oracle then by the birthday paradox bound:

$$|\ \Pr[\mathbf{G}_{0.2}] - \Pr[\mathbf{G}_{0.1}]\ | \leq q_{\mathtt{E}_1}^2 \cdot 2^{-\lambda_p - 1}$$

Game $\mathbf{G}_{0.3}$*:* This game handles the simulation of the second ideal cipher for the encryption of the ciphertext c. Equally to the previous game, $\mathcal{S}$ uses a list Λ_2. Λ_2 is a list of items $(\mathsf{ssid}, \mathsf{pw}, c, K, \mathtt{E}_2 \vee \mathtt{D}_2, \mathbf{Ec})$, even if the component K will only appear later. $\mathcal{S}$ simulates $\mathtt{E}_2$ and $\mathtt{D}_2$ as follows:

- On $\mathtt{E}_2(\mathsf{ssid}\|\mathsf{pw}, c)$:
  - If there exists a record $(\mathsf{ssid}, \mathsf{pw}, c, \star, \star, \mathbf{Ec}) \in \Lambda_2$ then $\mathcal{S}$ returns $\mathbf{Ec}$.
  - Else, $\mathcal{S}$ samples $\mathbf{Ec} \leftarrow\!\!\$\ \mathcal{C}'$. If $\mathbf{Ec}$ already exists in Λ_2, $\mathcal{S}$ aborts, else $\mathcal{S}$ records $(\mathsf{ssid}, \mathsf{pw}, c, \perp, \mathtt{E}_2, \mathbf{Ec})$ in Λ_2 and returns $\mathbf{Ec}$.
- On $\mathtt{D}_2(\mathsf{ssid}\|\mathsf{pw}, \mathbf{Ec})$:
  - If there is a record $(\mathsf{ssid}, \mathsf{pw}, c, \star, \star, \mathbf{Ec}) \in \Lambda_2$, $\mathcal{S}$ returns c.
  - Else, $\mathcal{S}$ samples $c \leftarrow\!\!\$\ \mathcal{C}$, records $(\mathsf{ssid}, \mathsf{pw}, c, \perp, \mathtt{D}_2, \mathbf{Ec})$ in Λ_2, and returns c.

Analysis: Similarly to the simulation of $\mathtt{E}_1$, $\mathcal{Z}$ is able to distinguish this simulation from the previous game if and only if $\mathcal{S}$ aborts. Assuming the cardinal of $\mathcal{C}$ is 2^{λ_c}, and if $\mathcal{A}$ makes up to $q_{\mathtt{E}_2}$ queries to the encryption oracle then by the birthday paradox bound:

$$|\ \Pr[\mathbf{G}_{0.3}] - \Pr[\mathbf{G}_{0.2}]\ | \leq q_{\mathtt{E}_2}^2 \cdot 2^{-\lambda_c - 1}$$

Eventually, as $\mathbf{G}_1 = \mathbf{G}_{0.3}$, $|\ \Pr[\mathbf{G}_1] - \Pr[\mathbf{G}_0]\ | \leq q_{\mathtt{E}_1}^2 \cdot 2^{-\lambda_p - 1} + q_{\mathtt{E}_2}^2 \cdot 2^{-\lambda_c - 1}$.

Game $\mathbf{G}_2$: Embedding of the Secrets. In this game we embed the associated secret keys in the simulation of both $\mathtt{D}_1$ and $\mathtt{D}_2$. We do it in two steps, first dealing with $\mathtt{D}_1$ in $\mathbf{G}_{1.1}$ and then with $\mathtt{D}_2$ in $\mathbf{G}_2 = \mathbf{G}_{1.2}$:

Game $\mathbf{G}_{1.1}$*:* We change the simulation of $\mathtt{D}_1$ and introduce the component sk in the records in Λ_1:

– On $\mathtt{D}_1(\mathsf{ssid}\|\mathsf{pw}, \mathbf{Epk})$:
  - If there is a record $(\mathsf{ssid}, \mathsf{pw}, pk, \star, \star, \mathbf{Epk}) \in \Lambda_1$, $\mathcal{S}$ returns pk.
  - Else, $\mathcal{S}$ builds $(pk, sk) \leftarrow \mathtt{KeyGen}(1^\kappa)$, records $(\mathsf{ssid}, \mathsf{pw}, pk, sk, \mathtt{D}_1, \mathbf{Epk})$ in Λ_1, and returns pk.

Analysis: The unique difference is a real public key instead of a random public key, which is exactly the fuzziness of the KEM, that we apply $q'_{\mathtt{D}_1}$ times in a hybrid sequence of games:

$$|\Pr[\mathbf{G}_{1.1}] - \Pr[\mathbf{G}_1]| \le q'_{\mathtt{D}_1} \cdot \mathsf{Adv}^{\text{fuzzy}}_{\mathtt{KEM}}(t)$$

We stress that $q'_{\mathtt{D}_1}$ will be the number of all the queries to $\mathtt{D}_1$ done by the simulator and by the adversary. This might be larger than the sole number $q_{\mathtt{D}_1}$ of queries asked by the adversary.

Game $\mathbf{G}_{1.2}$*:* Similarly to the previous subgame we change the simulation of $\mathtt{D}_2$, and introduce the component K in the records in Λ_2, but only for specific decryption calls by the simulator itself, with an additional input pk, that has necessarily been generated during the above simulation of $\mathtt{D}_1$:

– On $\mathtt{D}_2^*(\mathsf{ssid}\|\mathsf{pw}, \mathbf{Ec}, pk)$, by $\mathcal{S}$:
  - If there is a record $(\mathsf{ssid}, \mathsf{pw}, c, \star, \star, \mathbf{Ec}) \in \Lambda_2$, $\mathcal{S}$ returns c.
  - If there is no record $(\mathsf{ssid}, \mathsf{pw}, pk, \star, \mathtt{D}_1, \star) \in \Lambda_1$, $\mathcal{S}$ aborts.
  - Else, $\mathcal{S}$ builds $(c, K) \leftarrow \mathtt{Encaps}(pk)$, records $(\mathsf{ssid}, \mathsf{pw}, c, K, \mathtt{D}_2, \mathbf{Ec})$ in Λ_2, and returns c.

– On $\mathtt{D}_2(\mathsf{ssid}\|\mathsf{pw}, \mathbf{Ec})$, by $\mathcal{A}$, there is no change:
  - If there is a record $(\mathsf{ssid}, \mathsf{pw}, c, \star, \star, \mathbf{Ec}) \in \Lambda_2$, $\mathcal{S}$ returns c.
  - Else, $\mathcal{S}$ samples $c \leftarrow\!\!\$\, \mathcal{C}$, records $(\mathsf{ssid}, \mathsf{pw}, c, \perp, \mathtt{D}_2, \mathbf{Ec})$ in Λ_2, and returns c.

We will have to make sure the simulation never aborts here, with pk randomly generated (not under control of the adversary), without needing sk.

Analysis: As above, the unique difference is a real ciphertext instead of a random ciphertext, which is exactly the anonymity of the KEM, that we apply $q^*_{\mathtt{D}_2}$ times (the number of explicit $\mathtt{D}_2^*$ queries by the simulator) in a hybrid sequence of games, by guessing the good $\mathtt{D}_1$-query for pk:

$$|\Pr[\mathbf{G}_{1.2}] - \Pr[\mathbf{G}_{1.1}]| \le q_{\mathtt{D}_1} \cdot q^*_{\mathtt{D}_2} \cdot \mathsf{Adv}^{\text{ano}}_{\mathtt{KEM}}(t)$$

As $\mathbf{G}_2 = \mathbf{G}_{1.2}$, $|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]| \le q'_{\mathtt{D}_1} \cdot \mathsf{Adv}^{\text{fuzzy}}_{\mathtt{KEM}}(t) + q_{\mathtt{D}_1} \cdot q^*_{\mathtt{D}_2} \cdot \mathsf{Adv}^{\text{ano}}_{\mathtt{KEM}}(t)$.

Game $\mathbf{G_3}$: Simulation of Alice's Initialization. In this game, $\mathcal{S}$ simulates the first flow of Alice, using $\mathtt{D}_1$ instead of $\mathtt{E}_1$: it samples $\mathbf{Epk} \leftarrow\!\!\$\ \mathcal{P}'$, asks for $pk \leftarrow \mathtt{D}_1(\mathsf{ssid}\|\mathsf{pw}_A, \mathbf{Epk})$, which also generates sk, and sends $\mathbf{Epk}$ to Bob. This makes no difference from the previous game: $|\ \Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]\ | = 0$.

Game $\mathbf{G_4}$: Simulation of Bob's Answer. In this game, $\mathcal{S}$ simulates the second flow from an honest Bob, upon receiving $\mathbf{Epk}$. The behavior of $\mathcal{S}$ depends on the origin of the message $\mathbf{Epk}$: whether it comes from a honest Alice, or from the adversary $\mathcal{A}$, that has corrupted, or not, Alice. We thus do it in two steps, from $\mathbf{G}_3$, with $\mathbf{G}_{3.1}$ that deals with honestly generated $\mathbf{Epk}$, and $\mathbf{G}_4 = \mathbf{G}_{3.2}$ that deals with adversarially generated $\mathbf{Epk}$.

Game $\mathbf{G}_{3.1}$*:* $\mathbf{Epk}$ *comes from Alice.* $\mathbf{Epk}$ comes from the above simulation, with $(\mathsf{pw}_A, pk, sk, \mathtt{D}_1, \mathbf{Epk})$ in Λ_1. $\mathcal{S}$ asks for $pk' \leftarrow \mathtt{D}_1(\mathsf{ssid}\|\mathsf{pw}_B, \mathbf{Epk})$, which is either pk if the passwords are the same, or another pk', with associated sk'. $\mathcal{S}$ samples $\mathbf{Ec} \leftarrow\!\!\$\ \mathcal{C}'$, asks for $c \leftarrow \mathtt{D}_2^*(\mathsf{ssid}\|\mathsf{pw}_B, \mathbf{Ec}, pk')$, computes $K \leftarrow \mathtt{Decaps}(sk', c)$, and sends $\mathbf{Ec}$ to Alice. This makes no difference from the previous game, as pk' really comes from $\mathtt{D}_1$.

Game $\mathbf{G}_{3.2}$*:* $\mathbf{Epk}$ *comes from* $\mathcal{A}$*.* From the uniqueness of $\mathbf{Epk}$ in Λ_1, from explicit encryption $\mathtt{E}_1$ (or no record at all), $\mathcal{S}$ can extract at most one pair (pw, pk) used by $\mathcal{A}$:

– If $\mathsf{pw} = \mathsf{pw}_B$, with pk: $\mathcal{S}$ continues as Bob would do. It builds $(c, K) \leftarrow \mathtt{Encaps}(pk)$, $\mathbf{Ec} \leftarrow \mathtt{E}_1(\mathsf{ssid}\|\mathsf{pw}_B, c)$, and $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$.
– Else ($\mathsf{pw} \neq \mathsf{pw}_B$, or $\mathsf{pw} = \perp$): $\mathcal{S}$ asks for $pk \leftarrow \mathtt{D}_1(\mathsf{ssid}\|\mathsf{pw}_B, \mathbf{Epk})$, with sk, samples $\mathbf{Ec} \leftarrow \mathcal{C}'$, asks for $c \leftarrow \mathtt{D}_2^*(\mathsf{ssid}\|\mathsf{pw}_B, \mathbf{Ec}, pk)$, computes $K \leftarrow \mathtt{Decaps}(sk, c)$, and $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$.

This makes no difference from the previous game. As this last game $\mathbf{G}_{3.2}$ is $\mathbf{G}_4$, $|\ \Pr[\mathbf{G}_4] - \Pr[\mathbf{G}_3]\ | = 0$.

Game $\mathbf{G_5}$: Preparation of Alice's Reaction. We create a new list $\Lambda_{\mathbf{Epk}}$, with records of the form $(\mathsf{ssid}, \mathbf{Epk})$, initialized as an empty list. We first update the simulation of the first flow of Alice: for a new session with ssid, if there is a record $(\mathsf{ssid}, \mathbf{Epk}) \in \Lambda_{\mathbf{Epk}}$, $\mathcal{S}$ uses this $\mathbf{Epk}$, otherwise it samples an $\mathbf{Epk} \leftarrow\!\!\$\ \mathcal{P}'$, and adds $(\mathsf{ssid}, \mathbf{Epk})$ to $\Lambda_{\mathbf{Epk}}$.

For any query $\mathtt{D}_2(\mathsf{ssid}\|\mathsf{pw}, \mathbf{Ec})$ asked by the adversary or the simulator:

– If there is a record $(\mathsf{ssid}, \mathsf{pw}, c, \star, \star, \mathbf{Ec}) \in \Lambda_2$, $\mathcal{S}$ returns c.
– If there is a record $(\mathsf{ssid}, \mathbf{Epk}) \in \Lambda_{\mathbf{Epk}}$ it uses $\mathbf{Epk}$, otherwise it samples an $\mathbf{Epk} \leftarrow\!\!\$\ \mathcal{P}'$, and adds $(\mathsf{ssid}, \mathbf{Epk})$ to $\Lambda_{\mathbf{Epk}}$. Then, $\mathcal{S}$ first asks for $pk' \leftarrow \mathtt{D}_1(\mathsf{ssid}\|\mathsf{pw}, \mathbf{Epk})$, with sk', and then asks for $\mathtt{D}_2^*(\mathsf{ssid}\|\mathsf{pw}, \mathbf{Ec}, pk')$ with K'.

This makes no difference from the previous game, if we take care of the additional $\mathtt{D}_1$ queries, as $\mathbf{Epk}$ is still randomly sampled in $\mathcal{P}'$: $|\ \Pr[\mathbf{G}_5] - \Pr[\mathbf{G}_4]\ | = 0$.

Game $\mathbf{G_6}$: Simulation of Alice's Reaction. In this game, $\mathcal{S}$ simulates the key computation by Alice, upon receiving $\mathbf{Ec}$, which has been sent by either an honest Bob or the adversary. $\mathcal{S}$ first recovers Alice's secret key sk generated during the first honest flow. Then, we proceed in two steps, from $\mathbf{G}_5$, with $\mathbf{G}_{5.1}$ that deals with honestly generated $\mathbf{Ec}$, and $\mathbf{G}_6 = \mathbf{G}_{5.2}$ that deals with adversarially generated $\mathbf{Ec}$.

Game $\mathbf{G}_{5.1}$*:* $\mathbf{Ec}$ *comes from Bob.* We thus have $(\mathsf{ssid}, \mathsf{pw}, c, K, \mathtt{D}_2, \mathbf{Ec})$ in Λ_2. If $\mathsf{pw} = \mathsf{pw}_A$, we have both equalities $c = c' \leftarrow \mathtt{D}_2(\mathsf{ssid}\|\mathsf{pw}_A, \mathbf{Ec})$ and $K = K' \leftarrow \mathtt{Decaps}(sk, c')$: then Alice and Bob have the same final session keys $\mathtt{SK} = \mathtt{SK}' \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K')$. If $\mathsf{pw} \neq \mathsf{pw}_A$, we have both inequalities (excepted by chance) $c \neq c' \leftarrow \mathtt{D}_2(\mathsf{ssid}\|\mathsf{pw}_A, \mathbf{Ec})$ and $K \neq K' \leftarrow \mathtt{Decaps}(sk, c')$: then $\mathtt{SK}' \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K')$ is independent from the $\mathtt{SK}$ computed by Bob, excepted with random equality, which makes no difference from the previous game.

Game $\mathbf{G}_{5.2}$*:* $\mathbf{Ec}$ *comes from* $\mathcal{A}$*.* From the uniqueness of $\mathbf{Ec}$ in Λ_2, from explicit encryption $\mathtt{E}_2$ (or no record at all), $\mathcal{S}$ can extract at most one pair (pw, c) used by $\mathcal{A}$:

- If $\mathsf{pw} = \mathsf{pw}_A$, with c: $\mathcal{S}$ computes $K' \leftarrow \mathtt{Decaps}(sk, c)$ as well as the session key $\mathtt{SK}' \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K')$.
- Else ($\mathsf{pw} \neq \mathsf{pw}_A$, or $\mathsf{pw} = \perp$): $\mathcal{S}$ asks for $c' \leftarrow \mathtt{D}_2(\mathsf{ssid}\|\mathsf{pw}_A, \mathbf{Ec})$, computes $K' \leftarrow \mathtt{Decaps}(sk, c')$, and gets $\mathtt{SK}' \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K')$.

We stress that we use the above simulation of $\mathtt{D}_2$. This makes no difference from the previous game. As this last game $\mathbf{G}_{5.2}$ is $\mathbf{G}_6$, $|\Pr[\mathbf{G}_6] - \Pr[\mathbf{G}_5]| = 0$.

Game $\mathbf{G_7}$: Random Session Keys. Thanks to the above simulation of the ideal ciphers, $\mathcal{S}$ has the ability to extract the tentative password used by the adversary. It will be given access to two boolean functions;

- $\mathtt{GoodPwd}$ with input $(\mathsf{ssid}, P_i, \mathsf{pw})$ that answers whether this is the correct password of party P_i.
- $\mathtt{SamePwd}$ with input $(\mathsf{ssid}, P_i, P_j)$ that answers whether P_i and P_j share the same password.

We first replace session key K generation, for honest players, without knowing the passwords, by using $\mathtt{SamePwd}$ when it did not extract the password and $\mathtt{GoodPwd}$ when it successfully extracted pw using a private random oracle $\mathtt{H}^*_K$ onto $\mathcal{K}$, in $\mathbf{G}_{6.1}$. We then replace the final session key $\mathtt{SK}$ generation in $\mathbf{G}_7 = \mathbf{G}_{6.2}$.

Game $\mathbf{G}_{6.1}$*: Random* K'*.* We first use a private random oracle $\mathtt{H}^*_K$ onto $\mathcal{K}$ to replace K by random K', in some situations:

On Bob's Side: Upon receiving $\mathbf{Epk}$

- From an honest Alice, instead of setting $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$, if $\mathtt{SamePwd}(\mathsf{ssid}, P_i, P_j) = \mathsf{true}$, one sets $K' \leftarrow \mathtt{H}^*_K(\mathsf{ssid}, \mathsf{success})$ otherwise one sets $K' \leftarrow \mathtt{H}^*_K(\mathsf{ssid}, \mathsf{fail}_B)$, and updates the definition $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K')$.
- From $\mathcal{A}$, with extracted password pw: if $\mathtt{GoodPwd}(\mathsf{ssid}, P_j, \mathsf{pw}) = \mathsf{true}$, one keeps $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$; else one sets $K' \leftarrow \mathtt{H}^*_K(\mathsf{ssid}, \mathsf{fail}_B)$, and updates the definition $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K')$.

On Alice's Side: Upon receiving **Ec**

- From an honest Bob, instead of setting $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$, if $\mathtt{SamePwd}(\mathsf{ssid}, P_i, P_j) = \mathsf{true}$, one sets $K' \leftarrow \mathtt{H}^*_K(\mathsf{ssid}, \mathsf{success})$ otherwise one sets $K' \leftarrow \mathtt{H}^*_K(\mathsf{ssid}, \mathsf{fail}_A)$, and updates the definition $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K')$.
- From $\mathcal{A}$, with extracted password pw: if $\mathtt{GoodPwd}(\mathsf{ssid}, P_i, \mathsf{pw}) = \mathsf{true}$, one keeps $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K')$; else sets $K' \leftarrow \mathtt{H}^*_K(\mathsf{ssid}, \mathsf{fail}_A)$, and updates the definition $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K')$.

Analysis: We replace real keys K by random independent keys K', excepted when interacting with the adversary that has guessed the password. In all the modified sessions, K has been generated from a fresh KEM instance: From an honest Alice, Bob always uses a pk coming from $\mathtt{D}_1$ and a c coming from $\mathtt{D}_2^*$; if this comes from $\mathcal{A}$, unless the password was correctly guessed, pk also comes from $\mathtt{D}_1$ and c from $\mathtt{D}_2^*$. On Alice's side, unless the password was correctly guessed by the adversary, pk also comes from $\mathtt{D}_1$ and c from $\mathtt{D}_2^*$, thanks to the list $\Lambda_{\mathbf{Epk}}$ that prepared **Epk** in advance for any session. We can thus proceed with a sequence of hybrid games, replacing real keys by random keys.

- On Bob's side, the pairs come from $\mathtt{D}_1$ and $\mathtt{D}_2^*$, with known (pk, sk) and (c, K), with a unique call $\mathtt{D}_2^*$ per session: we can simply successively replace (pk, c, K) by (pk, c, K'), using the indistinguishability of the KEM: the gap is bounded by $q'_{\mathtt{D}_1} \cdot \mathsf{Adv}^{\mathrm{ind}}_{\mathtt{KEM}}(t)$.
- On Alice's side, the pairs come from $\mathtt{D}_1$ and $\mathtt{D}_2^*$, with known (pk, sk) and (c, K), but will have multiple generations per session: essentially, for each query $\mathtt{D}_2$, such a tuple must be created. We can successively replace (pk, c, K) by (pk, c, K'), using the indistinguishability of the KEM: the gap is bounded by $q'_{\mathtt{D}_2} \cdot \mathsf{Adv}^{\mathrm{ind}}_{\mathtt{KEM}}(t)$, where $q'_{\mathtt{D}_2}$ is the number of all the queries to $\mathtt{D}_2$ asked by the simulator and by the adversary.

We thus have $\mid \Pr[\mathbf{G}_{6.1}] - \Pr[\mathbf{G}_6] \mid \leq (q'_{\mathtt{D}_1} + q'_{\mathtt{D}_2}) \cdot \mathsf{Adv}^{\mathrm{ind}}_{\mathtt{KEM}}(t)$.

Game $\mathbf{G}_{6.2}$*: Random* $\mathtt{SK}$. We now replace $\mathtt{SK}$ by random $\mathtt{SK}'$, in some situations:

On Bob's Side: Upon receiving **Epk**

- From an honest Alice, instead of setting $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$, if $\mathtt{SamePwd}(\mathsf{ssid}, P_i, P_j) = \mathsf{true}$, one updates the generation by $\mathtt{SK} \leftarrow \mathtt{H}^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathsf{success})$, otherwise one uses the generation $\mathtt{SK} \leftarrow \mathtt{H}^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathsf{fail}_B)$.

  - From $\mathcal{A}$, with extracted password pw: if $\texttt{GoodPwd}(\mathsf{ssid}, P_j, \mathsf{pw}) = \mathsf{true}$, one keeps $\texttt{SK} \leftarrow \texttt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$; else one uses the generation $\texttt{SK} \leftarrow \texttt{H}^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathsf{fail}_B)$.

On Alice's Side: Upon receiving $\mathbf{Ec}$

  - From an honest Bob, instead of setting $\texttt{SK} \leftarrow \texttt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$, if $\texttt{SamePwd}(\mathsf{ssid}, P_i, P_j) = \mathsf{true}$, one updates the generation by $\texttt{SK} \leftarrow \texttt{H}^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathsf{success})$, otherwise one uses the generation $\texttt{SK} \leftarrow \texttt{H}^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathsf{fail}_1)$.
  - From $\mathcal{A}$, with extracted password pw: if $\texttt{GoodPwd}(\mathsf{ssid}, P_i, \mathsf{pw}) = \mathsf{true}$, one keeps $\texttt{SK} \leftarrow \texttt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K')$; else one uses the generation $\texttt{SK} \leftarrow \texttt{H}^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathsf{fail}_A)$.

The only way for the environment to detect the difference is to have a call $\texttt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$ that has been replaced by a call to $\texttt{H}^*$. But in the previous game, all the K that are in such changes are truly random, and there are at most $2q_s$ such changes, where q_s is the number of sessions: We thus have $| \Pr[\mathbf{G}_{6.2}] - \Pr[\mathbf{G}_{6.1}] | \leq q_{\texttt{H}} \cdot q_s / 2^{\lambda_k}$, where λ_k is the length of K. As this last game $\mathbf{G}_{6.2}$ is $\mathbf{G}_7$, $| \Pr[\mathbf{G}_7] - \Pr[\mathbf{G}_6] | \leq (q'_{\texttt{D}_1} + q'_{\texttt{D}_2}) \cdot \mathsf{Adv}^{\mathrm{ind}}_{\texttt{KEM}}(t) + q_{\texttt{H}} \cdot q_s / 2^{\lambda_k}$.

Game $\mathbf{G_8}$: Adaptive Corruptions. Since the values K and K' are not needed anymore to generate $\texttt{SK}$, we can postpone some evaluations that need the passwords of honest players, when corruptions happen:

- Alice's initialization: $\mathcal{S}$ uses or adds $\mathbf{Epk}$ in $\Lambda_{\mathbf{Epk}}$ and sends it. We postpone the evaluation of $pk \leftarrow \texttt{D}_1(\mathsf{ssid}\|\mathsf{pw}_A, \mathbf{Epk})$ with sk, at corruption time, to provide sk.
- Bob's answer to honest $\mathbf{Epk}$: $\mathcal{S}$ samples $\mathbf{Ec} \leftarrow\!\$\ \mathcal{C}'$ and sends it. We postpone the evaluations $c \leftarrow \texttt{D}_2^*(\mathsf{ssid}\|\mathsf{pw}_B, \mathbf{Ec}, pk)$ and $K \leftarrow \texttt{Decaps}(sk, c)$, at corruption time, to provide K. The evaluation of $\texttt{SK}$ uses $\texttt{H}^*$, with inputs depending on similar or different passwords.
- Alice's reaction to honest $\mathbf{Ec}$: The evaluation of $\texttt{SK}$ uses $\texttt{H}^*$, with inputs depending on similar or different passwords.

In case of late corruptions, after $\texttt{SK}$ has been set, from the knowledge of the passwords, one can program the random oracle $\texttt{H}$ to make it consistent with K' obtained from $\texttt{H}^*$:

- if Alice is corrupted, from pw_A, for all the $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathsf{status}, \texttt{SK})$ in $\Lambda_{\texttt{H}^*}$, involving Alice as P_i, one queries $pk \leftarrow \texttt{D}_1(\mathsf{ssid}\|\mathsf{pw}_A, \mathbf{Epk})$ with sk, and $c \leftarrow \texttt{D}_2(\mathsf{ssid}\|\mathsf{pw}_A, \mathbf{Ec})$. This always succeeds as $\mathbf{Epk}$ was a fresh value. Then one can compute $K \leftarrow \texttt{Decaps}(sk, c)$ to add $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K, \texttt{SK})$ in $\Lambda_{\texttt{H}}$.
- if Bob is corrupted, from pw_B, for all the $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathsf{status}, \texttt{SK})$ in $\Lambda_{\texttt{H}^*}$, involving Bob as P_j, one queries $pk \leftarrow \texttt{D}_1(\mathsf{ssid}\|\mathsf{pw}_B, \mathbf{Epk})$ with sk, and $c \leftarrow \texttt{D}_2(\mathsf{ssid}\|\mathsf{pw}_B, \mathbf{Ec})$. This always succeeds as $\mathbf{Epk}$ has not been obtained as an encryption under $\texttt{E}_1$ (otherwise $\texttt{SK}$ has already been generated with $\texttt{H}$). Then one computes $K \leftarrow \texttt{Decaps}(sk, c)$ to add $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K, \texttt{SK})$ in $\Lambda_{\texttt{H}}$.

In the previous game, we have already excluded queries on these specific inputs, so all these programmings are possible: $|\Pr[\mathbf{G}_8] - \Pr[\mathbf{G}_7]| = 0$.

Game $\mathbf{G_9}$: Adding the Full $\mathcal{F}_{pwKE}$ Interface. In this game we add the full $\mathcal{F}_{pwKE}$ interface to fully model the ideal world. First, $\mathcal{S}$ simulates its use of GoodPwd by querying TestPwd to $\mathcal{F}_{pwKE}$ on input $(\mathsf{ssid}, P_i, \mathsf{pw})$ to test if pw is the password associated to P_i in ssid. Then, for key generation, when a value K can be computed, and $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$, then one queries $\mathcal{F}_{pwKE}$ on $(\mathtt{NewKey}, \mathsf{ssid}, P, \mathtt{SK})$, for any party P.

Let us show this provides the same output as in the previous game: First, in the simulation, if $\mathcal{S}$ has extracted the password used by the adversary, then a TestPwd query has been sent to $\mathcal{F}_{pwKE}$. According to the ideal functionality two cases arise, according to the correct guess:

- If the guess is incorrect: the record is marked as interrupted.
- If the guess is correct: the record is marked as compromised.

If the session is compromised, $\mathtt{NewKey}(\mathsf{ssid}, P_i, \mathtt{SK})$ returns SK to P_i, which are the cases where we kept the definition of SK with H. If the session is interrupted it returns a random $\mathtt{SK}'$ to P_i, which are the cases where we used $\mathtt{H}^*$ with fail. This behavior of NewKey is exactly the one from that has been simulated by $\mathcal{S}$ and therefore remains indistinguishable.

Now assuming that $\mathcal{S}$ could not extract a password: it has sent random flow **Epk** and **Ec** and used its private oracle $\mathtt{H}^*$ to build a random session key for each party. Since $\mathcal{S}$ does not know the parties' passwords it is not able to tell if it needs to derive the same session key for both of them. Now, even though $\mathcal{S}$ derived the same session key SK using $\mathtt{H}^*$ on behalf of Alice and Bob, by definition of the NewKey interface:

- Two honest parties in a fresh session, using the same password, derive the same session key, because of a success status.
- Two honest parties in a fresh session, not using the same passwords, derive two random different session keys, because of the fail status.

Laslty on NewSession, $\mathcal{S}$ does nothing more than what is described in the simulation. Only $\mathcal{F}_{pwKE}$ does his internal computation.
Hence, we have $|\Pr[\mathbf{G}_9] - \Pr[\mathbf{G}_8]| = 0$, and this game is perfectly indistinguishable from the ideal world. Note that now the private oracle H does not need its status component anymore because it is handled by $\mathcal{F}_{pwKE}$ entirely.

The global gap is thus:

$$\begin{aligned} |\Pr[\mathbf{G}_9] - \Pr[\mathbf{G}_0]| \leq\ & q_{\mathtt{E}_1}^2 \cdot 2^{-\lambda_p - 1} + q_{\mathtt{E}_2}^2 \cdot 2^{-\lambda_c - 1} \\ & + q'_{\mathtt{D}_1} \cdot \mathsf{Adv}_{\mathtt{KEM}}^{\text{fuzzy}}(t) + q_{\mathtt{D}_1} \cdot q^*_{\mathtt{D}_2} \cdot \mathsf{Adv}_{\mathtt{KEM}}^{\text{ano}}(t) \\ & + (q'_{\mathtt{D}_1} + q'_{\mathtt{D}_2}) \cdot \mathsf{Adv}_{\mathtt{KEM}}^{\text{ind}}(t) + q_{\mathtt{H}} \cdot q_s / 2^{\lambda_k} \end{aligned}$$

where we denote the global numbers of queries asked by the adversary $q_{\mathtt{E}_1}$, $q_{\mathtt{D}_1}$, $q_{\mathtt{E}_2}$, $q_{\mathtt{D}_2}$, and $q_{\mathtt{H}}$. But we also need to count the number of queries to $\mathtt{D}_1$, $\mathtt{D}_2$ and $\mathtt{D}_2^*$ by the adversary and the simulator, at some point of the simulation:

- the global number of queries asked to $\mathtt{D}_1$ is q_s (for initialization by Alice or answer by Bob) plus $q_{\mathtt{D}_1}$, and $q_{\mathtt{D}_2}$, as all the queries to $\mathtt{D}_2$ make a call to $\mathtt{D}_1$;
- the global number of queries asked to $\mathtt{D}_2$ is q_s (for answer by Bob) plus $q_{\mathtt{D}_2}$;
- the global number of queries asked to $\mathtt{D}_2^*$ is q_s (for answer by Bob) plus $q_{\mathtt{D}_2}$.

Hence,

$$\begin{aligned}
|\Pr[\mathbf{G}_9] - \Pr[\mathbf{G}_0]| \le\ & q_{\mathtt{E}_1}^2 \cdot 2^{-\lambda_p - 1} + q_{\mathtt{E}_2}^2 \cdot 2^{-\lambda_c - 1} \\
& + (q_s + q_{\mathtt{D}_1}) \cdot \mathsf{Adv}_{\mathtt{KEM}}^{\text{fuzzy}}(t) + q_{\mathtt{D}_1} \cdot (q_s + q_{\mathtt{D}_2}) \cdot \mathsf{Adv}_{\mathtt{KEM}}^{\text{ano}}(t) \\
& + (2q_s + q_{\mathtt{D}_1} + q_{\mathtt{D}_2}) \cdot \mathsf{Adv}_{\mathtt{KEM}}^{\text{ind}}(t) + q_{\mathtt{H}} \cdot q_s / 2^{\lambda_k}
\end{aligned}$$

```
On $\mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$
If $\exists(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K, \mathtt{SK}) \in \Lambda_{\mathtt{H}}$
    return $\mathtt{SK}$
Else:
    sample $\mathtt{SK} \leftarrow\$ \{0,1\}^{\lambda_{\mathtt{H}}}$ and record
    $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K, \mathtt{SK}) \in \Lambda_{\mathtt{H}}$
    return $\mathtt{SK}$

On $\mathtt{H}^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec})$
If $\exists(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathtt{SK}) \in \Lambda_{\mathtt{H}^*}$
    return $\mathtt{SK}$
Else:
    sample $\mathtt{SK} \leftarrow\$ \{0,1\}^{\lambda_{\mathtt{H}}}$ and record:
    $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathtt{SK}) \in \Lambda_{\mathtt{H}^*}$
    return $\mathtt{SK}$

On $\mathtt{E}_1(\mathsf{ssid}||\mathsf{pw}, pk)$
If $\exists(\mathsf{ssid}, \mathsf{pw}, pk, \star, \star, \mathbf{Epk}) \in \Lambda_1$ :
    returns $\mathbf{Epk}$.
Otherwise: $\mathbf{Epk} \leftarrow\$ \mathcal{P}'$.
If: $\exists(\star, \star, \star, \star, \star, \mathbf{Epk}) \in \Lambda_1$ :
    $\mathcal{S}$ aborts.
Else: $\mathcal{S}$ records :
    $(\mathsf{ssid}, \mathsf{pw}, pk, \perp, \mathtt{E}_1, \mathbf{Epk}) \in \Lambda_1$
    returns $\mathbf{Epk}$.

On $\mathtt{D}_1(\mathsf{ssid}||\mathsf{pw}, \mathbf{Epk})$
If $\exists(\mathsf{ssid}, \mathsf{pw}, pk, \star, \star, \mathbf{Epk}) \in \Lambda_1$: returns $pk$.
Else, $\mathcal{S}$ builds $(pk, sk) \leftarrow \mathtt{KeyGen}(1^\kappa)$
records $(\mathsf{ssid}, \mathsf{pw}, pk, sk, \mathtt{D}_1, \mathbf{Epk}) \in \Lambda_1$
returns $pk$.

On $\mathtt{E}_2(\mathsf{ssid}||\mathsf{pw}, c)$
If $\exists(\mathsf{ssid}, \mathsf{pw}, c, \star, \star, \mathbf{Ec}) \in \Lambda_2$:
    returns $\mathbf{Ec}$.
Otherwise: $\mathbf{Ec} \leftarrow\$ \mathcal{C}'$.
If: $\exists(\star, \star, \star, \star, \star, \mathbf{Ec}) \in \Lambda_2$:
    $\mathcal{S}$ aborts.
Else: $\mathcal{S}$ records :
    $(\mathsf{ssid}, \mathsf{pw}, c, \perp, \mathtt{E}_2, \mathbf{Ec}) \in \Lambda_2$,
    returns $\mathbf{Ec}$.

On $\mathtt{D}_2^*(\mathsf{ssid}||\mathsf{pw}, \mathbf{Ec})$ by $\mathcal{S}$
If $\exists(\mathsf{ssid}, \mathsf{pw}, c, \star, \star, \mathbf{Ec}) \in \Lambda_2$: returns $c$.
If $\nexists(\mathsf{ssid}, \mathsf{pw}, pk, \star, \mathtt{D}_1, \star) \in \Lambda_1$: $\mathcal{S}$ aborts.
Else, $\mathcal{S}$ builds $(c, K) \leftarrow \mathtt{Encaps}(pk)$,
records $(\mathsf{ssid}, \mathsf{pw}, c, K, \mathtt{D}_2, \mathbf{Ec}) \in \Lambda_2$, returns $c$.

On $\mathtt{D}_2(\mathsf{ssid}||\mathsf{pw}, \mathbf{Ec})$ by $\mathcal{A}$
If $\exists(\mathsf{ssid}, \mathsf{pw}, c, \star, \star, \mathbf{Ec}) \in \Lambda_2$: returns $c$.
Else, $\mathcal{S}$ samples $(c \leftarrow\$ \mathcal{C})$,
records: $(\mathsf{ssid}, \mathsf{pw}, c, \perp, \mathtt{D}_2, \mathbf{Ec}) \in \Lambda_2$, returns $c$.
```

Fig. 7. Simulation of the ideal ciphers and the random oracle in proof of Theorem 1.

```
On $(\texttt{NewSession}, \mathsf{ssid}, P_i, P_j)$ from $\mathcal{F}_{pwKE}$
If $P_i = \mathsf{client}$ :
    $\mathbf{Epk} \leftarrow_{\$} \mathcal{P}'$, sends $\mathbf{Epk}$

On $(\texttt{AdaptiveCorruption}, \mathsf{ssid}, P_i)$ from $\mathcal{Z}$
$\mathcal{S}$ gets: $(\mathsf{ssid}, P_i, \mathsf{pw}_i)$.
 $\rightarrow$ Before $\texttt{SK}$ is set:
$\mathsf{client}$: $pk \leftarrow \texttt{D}_1(\mathsf{ssid}||\mathsf{pw}_A, \mathbf{Epk})$
$\mathsf{server}$: gets $c \leftarrow \texttt{D}_2^*(\mathsf{ssid}||\mathsf{pw}_B, \mathbf{Ec}, pk)$ and $K \leftarrow \texttt{Decaps}(sk, c)$
$\texttt{SK} \leftarrow \texttt{H}^*$ depending on $\mathsf{pw}_A \overset{?}{=} \mathsf{pw}_B$
 $\rightarrow$ After $\texttt{SK}$ is set:
$\mathsf{client}$ gets:
    $pk \leftarrow \texttt{D}_1(\mathsf{ssid}||\mathsf{pw}_A, \mathbf{Epk})$, $c \leftarrow \texttt{D}_2^*(\mathsf{ssid}||\mathsf{pw}_A, \mathbf{Ec}, pk)$ and $K \leftarrow \texttt{Decaps}(sk, c)$
$\mathsf{server}$ gets:
    $pk \leftarrow \texttt{D}_1(\mathsf{ssid}||\mathsf{pw}_B, \mathbf{Epk})$, $c \leftarrow \texttt{D}_2(\mathsf{ssid}||\mathsf{pw}_B, \mathbf{Ec}, pk)$ and $K \leftarrow \texttt{Decaps}(sk, c)$
Record: $(\mathsf{ssid}, P_{\mathsf{client}}, P_{\mathsf{server}}, \mathbf{Epk}, \mathbf{Ec}, K, \texttt{SK}) \in \Lambda_{\texttt{H}}$
```

Fig. 8. Simulation of the behavior against $\mathcal{Z}$ and $\mathcal{F}_{pwKE}$ in proof of Theorem 1.

```
Upon server $P_i$ receiving $\mathbf{Epk}$
If $\exists(\mathsf{ssid}, \mathsf{pw}, pk, \star, \star, \mathbf{Epk})$:
  Upon positive answer after querying: $(\mathtt{TestPwd}, \mathsf{ssid}, P_i, \mathsf{pw})$ to $\mathcal{F}_{pwKE}$ :
    compute: $(c, K) \leftarrow \mathtt{Encaps}(pk)$, $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K, \mathtt{SK})$
Else:
  sample: $\mathbf{Ec} \leftarrow\!\!\$\; \mathcal{C}'$, $\mathtt{SK} \leftarrow \mathtt{H}^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec})$
send: $\mathbf{Ec}$ to $P_j$
send: $(\mathtt{NewKey}, \mathsf{ssid}, P_i, \mathtt{SK})$

Preparation of client $P_i$
On $\mathtt{D}_2(\mathsf{ssid}||\mathsf{pw}, \mathbf{Ec})$ from $\mathcal{A}$ or $\mathcal{S}$ without existing record in $\Lambda_2$:
If $\exists(\mathsf{ssid}, \mathbf{Epk}) \in \Lambda_{\mathbf{Epk}}$: return $\mathbf{Epk}$
Else:
  sample $\mathbf{Epk} \leftarrow\!\!\$\; \mathcal{P}'$, add $(\mathsf{ssid}, \mathbf{Epk})$ to $\Lambda_{\mathbf{Epk}}$
  get $pk' \leftarrow \mathtt{D}_1(\mathsf{ssid}||\mathsf{pw}, \mathbf{Epk})$ with $sk'$
  get $c \leftarrow \mathtt{D}_2^*(\mathsf{ssid}||\mathsf{pw}, \mathbf{Ec}, pk')$ and extract $K$

Upon client $P_i$ receiving $\mathbf{Ec}$
If $\exists(\mathsf{ssid}, \mathsf{pw}, c, \star, \star, \mathbf{Ec})$:
  Upon positive answer after querying: $(\mathtt{TestPwd}, \mathsf{ssid}, P_i, \mathsf{pw})$ to $\mathcal{F}_{pwKE}$ :
    get: $sk, pk \leftarrow \mathtt{D}_1(\mathsf{ssid}||\mathsf{pw}, \mathbf{Epk})$
    extract $K$ or compute: $K \leftarrow \mathtt{Decaps}(sk, c)$
    get: $\mathtt{SK} \leftarrow \mathtt{H}(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K, \mathtt{SK})$
Else:
  get: $\mathtt{SK} \leftarrow \mathtt{H}^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec})$
send: $(\mathtt{NewKey}, \mathsf{ssid}, P_i, \mathtt{SK})$
```

Fig. 9. Simulation of the protocol from the client and server point of view in proof of Theorem 1.

B Proof of Theorem 2

Game $\mathbf{G_0}$: In this game we present a formalization of the OCAKE protocol using the random oracle model and the ideal cipher model. Our simulation is set in the erasure model, in which secret information is erased from the memory of each party when it is no longer needed during the protocol execution. The protocol is executed in an adversarial environment, denoted as $\mathcal{Z}$, in which the parties can be statistically corrupted by an adversary $\mathcal{A}$ to leak their private state. In our simulation, Alice is referred to as P_i (the initiator) and Bob is referred to as P_j (the responder). Additionally, pw_A represents Alice's password and pw_B represents Bob's password, while pw represents an arbitrary password usually used by the adversary.

Game $\mathbf{G_1}$: Simulation of $\mathcal{F}_{IC}$ and $\mathcal{F}_{RO}$. This game builds the simulation from $\mathcal{S}$ of the different oracles namely: the ideal cipher and the random oracle. It is subdivisided into three subgames and each subgame represents respectively the simulation of two different random oracles and one ideal cipher, starting from $\mathbf{G}_0$, with $\mathbf{G}_{0.1}$, $\mathbf{G}_{0.2}$, and $\mathbf{G}_{0.3} = \mathbf{G}_1$:

Game $\mathbf{G}_{0.1}$*:* In this subgame, $\mathcal{S}$ models the random oracle $\mathtt{H}_1$ used for authentication, where on each query the oracle returns a uniformly random answer. We will also need to exclude collisions. To remain consistent with previous answers $\mathcal{S}$ uses a list $\Lambda_{\mathtt{H}_1}$ of tuples $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K, \mathtt{Auth})$. This list is initially set as empty, then on a query $\mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K)$, $\mathcal{S}$ uses $\Lambda_{\mathtt{H}_1}$ to simulate it as follows:

- If a record $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K, \mathtt{Auth})$ exists in $\Lambda_{\mathtt{H}_1}$, $\mathcal{S}$ returns $\mathtt{Auth}$.
- Else, $\mathcal{S}$ samples a random $\mathtt{Auth}$, if $\mathtt{Auth}$ already exists as a previous answer, $\mathcal{S}$ aborts, else $\mathcal{S}$ records $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K, \mathtt{Auth})$ in $\Lambda_{\mathtt{H}_1}$, and returns $\mathtt{Auth}$.

However throughout the simulation, $\mathcal{S}$ will have to simulate $\mathtt{H}_1$ while not knowing K nor pw, for generating $\mathtt{Auth}$. $\mathcal{S}$ uses a private oracle $\mathtt{H}_1^*$ to record values in a list $\Lambda_{\mathtt{H}_1^*}$ of items $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{status}, \mathtt{Auth})$. $\mathcal{S}$ simulates $\mathtt{H}_1^*$ as follows on input $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{status})$, where status can be $\mathsf{success}$ or fail, where $\mathsf{success}$ can lead to an accepted authentication with $\mathtt{Auth}$, whereas a failure will lead to a failed $\mathtt{Auth}$ meaning an abortion:

- If a record $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{status}, \mathtt{Auth})$ exists in $\Lambda_{\mathtt{H}_1^*}$, $\mathcal{S}$ returns $\mathtt{Auth}$.
- Else, $\mathcal{S}$ samples a random $\mathtt{Auth}$, if $\mathtt{Auth}$ already exists as a previous answer, $\mathcal{S}$ aborts, else $\mathcal{S}$, records $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{status}, \mathtt{Auth})$ in $\Lambda_{\mathtt{H}_1^*}$, and returns $\mathtt{Auth}$.

Additionally we set a query to $\mathtt{H}_1$ with $K = \perp$ to return special character $\emptyset$ to force the authentication fail.

Analysis: Under the assumption of the Random Oracle model depicted by its ideal functionality, $\mathcal{Z}$ can distinguish the real execution of the protocol from this game if $\mathcal{S}$ aborts. Assuming the output length of $\mathtt{H}_1$ and $\mathtt{H}_1^*$ to be $\lambda_{\mathtt{H}_1}$, and $q'_{\mathtt{H}_1}$ being the global number of queries to $\mathtt{H}_1$ and $\mathtt{H}_1^*$ (by both the simulator and the adversary), the birthday paradox gives:

$$|\Pr[\mathbf{G}_{0.1}] - \Pr[\mathbf{G}_0]| \leq {q'_{\mathtt{H}_1}}^2 \cdot 2^{-\lambda_{\mathtt{H}_1}-1}$$

Game $\mathbf{G}_{0.2}$*:* In this subgame, $\mathcal{S}$ models the random oracle $\mathtt{H}_2$ to build session keys, where on each query the oracle returns a uniformly random answer. To remain consistent with previous answers $\mathcal{S}$ uses a list $\Lambda_{\mathtt{H}_2}$ of tuples $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K, \mathtt{SK})$. This list is initially set as empty, and on a query $\mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K, \mathtt{SK})$, $\mathcal{S}$ uses $\Lambda_{\mathtt{H}_2}$ to simulate it as follows:

– If a record $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K, \mathtt{SK})$ exists in $\Lambda_{\mathtt{H}_2}$, $\mathcal{S}$ returns $\mathtt{SK}$.
– Else, $\mathcal{S}$ samples a random $\mathtt{SK}$, records $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K, \mathtt{SK})$ in $\Lambda_{\mathtt{H}_2}$, and returns $\mathtt{SK}$.

However throughout the simulation, $\mathcal{S}$ will have to simulate $\mathtt{H}_2$ while not knowing K, for generating $\mathtt{SK}$. $\mathcal{S}$ uses a private oracle $\mathtt{H}_2^*$ to record values in a list $\Lambda_{\mathtt{H}_2^*}$ of items $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, \mathsf{status}, \mathtt{SK})$. $\mathcal{S}$ simulates $\mathtt{H}_2^*$ as follows on input $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, \mathsf{status})$, where status can be $\mathsf{success}$, fail_A, or fail_B, where both $\mathsf{success}$ would lead to the same key $\mathtt{SK}$, whereas a failure will lead to independent keys:

– If a record $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, \mathsf{status}, \mathtt{SK})$ exists in $\Lambda_{\mathtt{H}_2^*}$, $\mathcal{S}$ returns $\mathtt{SK}$.
– Else, $\mathcal{S}$ samples a random $\mathtt{SK}$, records $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, \mathsf{status}, \mathtt{SK})$ in $\Lambda_{\mathtt{H}_2^*}$, and returns $\mathtt{SK}$.

This simulation is perfectly identical to $\mathbf{G}_{0.1}$.

Game $\mathbf{G}_{0.3}$*:* In this game, we simulate an ideal cipher from the first exchange of the protocol. The simulation of this oracle needs to be consistent, meaning that any query that has already been asked should return the same answer as the first time it was asked. Additionally, the simulation needs to capture the properties of an ideal cipher, which means simulating each encryption as a random bijection for each key (actually, for each $\mathsf{ssid}\|\mathsf{pw}$). But for the following simulation, we will need to avoid collisions during adversary's encryption with different inputs. Then, the simulator uses a list called $\Lambda_{\mathtt{E}}$ for encryption and decryption queries on each oracle. $\Lambda_{\mathtt{E}}$ is composed of tuples of the form $(\mathsf{ssid}, \mathsf{pw}, pk, \mathtt{E} \vee \mathtt{D}, \mathbf{Epk})$, even if the component sk will only appear later. $\mathcal{S}$ simulates $\mathtt{E}$ and $\mathtt{D}$ as follows:

– On $\mathtt{E}(\mathsf{ssid}\|\mathsf{pw}, pk)$:
  - If there exists a record $(\mathsf{ssid}, \mathsf{pw}, pk, \star, \star, \mathbf{Epk}) \in \Lambda_{\mathtt{E}}$ then $\mathcal{S}$ returns $\mathbf{Epk}$.
  - Else, $\mathcal{S}$ samples $\mathbf{Epk} \leftarrow\!\!\$\ \mathcal{P}'$. If $\mathbf{Epk}$ already exists in $\Lambda_{\mathtt{E}}$, $\mathcal{S}$ aborts, else $\mathcal{S}$ records $(\mathsf{ssid}, \mathsf{pw}, pk, \mathtt{E}, \mathbf{Epk})$ in $\Lambda_{\mathtt{E}}$ and returns $\mathbf{Epk}$.
– On $\mathtt{D}(\mathsf{ssid}\|\mathsf{pw}, \mathbf{Epk})$:
  - If there is a record $(\mathsf{ssid}, \mathsf{pw}, pk, \star, \star, \mathbf{Epk}) \in \Lambda_{\mathtt{E}}$, $\mathcal{S}$ returns pk.
  - Else, $\mathcal{S}$ samples $pk \leftarrow\!\!\$\ \mathcal{P}$, records $(\mathsf{ssid}, \mathsf{pw}, pk, \mathtt{D}, \mathbf{Epk})$ in $\Lambda_{\mathtt{E}}$, and returns pk.

Analysis: Under the assumption of the Ideal Cipher model depicted by its ideal functionality, $\mathcal{Z}$ can distinguish the real execution of the protocol from this game if $\mathcal{S}$ aborts. Assuming the cardinal of $\mathcal{P}$ is 2^{λ_p}, and if $\mathcal{A}$ makes up to $q_{\mathtt{E}}$ queries to the encryption oracle then by the birthday paradox bound:

$$|\Pr[\mathbf{G}_{0.3}] - \Pr[\mathbf{G}_{0.2}]| \leq q_{\mathtt{E}}^2 \cdot 2^{-\lambda_p - 1}$$

Eventually, as $\mathbf{G}_1 = \mathbf{G}_{0.3}$, $|\Pr[\mathbf{G}_1] - \Pr[\mathbf{G}_0]| \leq {q'_{\mathtt{H}_1}}^2 \cdot 2^{-\lambda_{\mathtt{H}_1} - 1} + q_{\mathtt{E}}^2 \cdot 2^{-\lambda_p - 1}$.

Game $\mathbf{G}_2$: Embedding of the Secrets. In this game we embed the associated secret keys in the simulation of $\mathtt{D}$ and introduce the component *sk* in the records in $\Lambda_{\mathtt{E}}$:

- On $\mathtt{D}_1(\mathsf{ssid}\|\mathsf{pw}, \mathbf{Epk})$:
  - If there is a record $(\mathsf{ssid}, \mathsf{pw}, pk, \star, \star, \mathbf{Epk}) \in \Lambda_{\mathtt{E}}$, $\mathcal{S}$ returns *pk*.
  - Else, $\mathcal{S}$ builds $(pk, sk) \leftarrow \mathtt{KeyGen}(1^\kappa)$, records $(\mathsf{ssid}, \mathsf{pw}, pk, sk, \mathtt{D}, \mathbf{Epk})$ in $\Lambda_{\mathtt{E}}$, and returns *pk*.

Analysis: The unique difference is a real public key instead of a random public key, which is exactly the fuzziness of the KEM, that we apply $q'_{\mathtt{D}_1}$ times in a hybrid sequence of games:

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]| \leq q'_{\mathtt{D}} \cdot \mathsf{Adv}_{\mathtt{KEM}}^{\mathrm{fuzzy}}(t)$$

We stress that $q'_{\mathtt{D}}$ will be the number of all the queries to $\mathtt{D}$ done by the simulator and by the adversary. This might be larger than the sole number $q_{\mathtt{D}}$ of queries asked by the adversary.

Game $\mathbf{G}_3$: $\boldsymbol{\mathcal{A}}$ randomly guessing *Auth*. In this game we model the capacity of the adversary to randomly guess $\mathtt{Auth}$, without asking the right query to $\mathtt{H}_1$. If such a case happens, $\mathcal{S}$ now aborts.

Analysis: In such a case, Alice asks a fresh query to $\mathtt{H}_1$ which provides a random answer:

$$|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]| \leq q_s \cdot 2^{-\lambda_{\mathtt{H}_1}}$$

Game $\mathbf{G}_4$: Simulation of Alice's Initialization. In this game, $\mathcal{S}$ simulates the first flow of Alice, using $\mathtt{D}$ instead of $\mathtt{E}$: it samples $\mathbf{Epk} \leftarrow\!\!\$\ \mathcal{P}'$, asks for $pk \leftarrow \mathtt{D}(\mathsf{ssid}\|\mathsf{pw}_A, \mathbf{Epk})$, which also generates *sk*, and sends $\mathbf{Epk}$ to Bob. This makes no difference from the previous game: $|\Pr[\mathbf{G}_4] - \Pr[\mathbf{G}_3]| = 0$.

Game $\mathbf{G}_5$: Simulation of Bob's Answer. In this game, $\mathcal{S}$ simulates the second flow from an honest Bob, upon receiving $\mathbf{Epk}$. The behavior of $\mathcal{S}$ depends on the origin of the message $\mathbf{Epk}$: whether it comes from a honest Alice, or from the adversary $\mathcal{A}$, that has corrupted, or not, Alice. We thus do it in two steps, from $\mathbf{G}_4$, with $\mathbf{G}_{4.1}$ that deals with honestly generated $\mathbf{Epk}$, and $\mathbf{G}_5 = \mathbf{G}_{4.2}$ that deals with adversarially generated $\mathbf{Epk}$.

Game $\mathbf{G}_{4.1}$*:* **Epk** *comes from Alice.* **Epk** comes from the above simulation, with $(\mathsf{pw}_A, pk, sk, \mathtt{D}, \mathbf{Epk})$ in $\Lambda_{\mathtt{E}}$. $\mathcal{S}$ asks for $pk' \leftarrow \mathtt{D}(\mathsf{ssid}\|\mathsf{pw}_B, \mathbf{Epk})$, which is either pk if the passwords are the same, or another pk', with associated sk'. $\mathcal{S}$ builds $c \leftarrow \mathtt{Encaps}(pk')$, computes $K \leftarrow \mathtt{Decaps}(sk', c)$, and sends c to Alice, together with $\mathtt{Auth} = \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}_B, K)$ This makes no difference from the previous game, as pk' really comes from $\mathtt{D}$.

Game $\mathbf{G}_{4.2}$*:* **Epk** *comes from* $\mathcal{A}$*.* From the uniqueness of **Epk** in $\Lambda_{\mathtt{E}}$, from explicit encryption $\mathtt{E}$ (or no record at all), $\mathcal{S}$ can extract at most one pair (pw, pk) used by $\mathcal{A}$:

- If $\mathsf{pw} = \mathsf{pw}_B$, with pk: $\mathcal{S}$ continues as Bob does, with $(c, K) \leftarrow \mathtt{Encaps}(pk)$, it gets $\mathtt{Auth} \leftarrow \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K)$, and builds accordingly $\mathtt{SK} \leftarrow \mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K)$.
- Else ($\mathsf{pw} \neq \mathsf{pw}_B$, or $\mathsf{pw} = \perp$): $\mathcal{S}$ asks for $pk \leftarrow \mathtt{D}(\mathsf{ssid}\|\mathsf{pw}_B, \mathbf{Epk})$, with sk, gets $(c, K) \leftarrow \mathtt{Encaps}(pk)$ and $\mathtt{Auth} \leftarrow \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathsf{pw}, \mathbf{Epk}, c, K)$, and $\mathtt{SK} \leftarrow \mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K)$.

This makes no difference from the previous game. As this last game $\mathbf{G}_{4.2}$ is $\mathbf{G}_5$, $|\Pr[\mathbf{G}_5] - \Pr[\mathbf{G}_4]| = 0$.

Game $\mathbf{G}_6$: Simulation of Alice's Reaction. In this game, $\mathcal{S}$ simulates the key computation by Alice, upon receiving $(c, \mathtt{Auth})$, which has been sent by either an honest Bob or the adversary. $\mathcal{S}$ first recovers Alice's secret key sk generated during the first honest flow. Then, we proceed in two steps, from $\mathbf{G}_5$, with $\mathbf{G}_{5.1}$ that deals with honestly generated $(c, \mathtt{Auth})$, and $\mathbf{G}_6 = \mathbf{G}_{5.2}$ that deals with adversarially generated $(c, \mathtt{Auth})$.

Game $\mathbf{G}_{5.1}$*:* $(c, \mathtt{Auth})$ *comes from Bob.* If $\mathsf{pw} = \mathsf{pw}_A$, we have both equalities $K = K' \leftarrow \mathtt{Decaps}(sk, c')$ and $\mathtt{Auth} = \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathsf{pw}, \mathbf{Epk}, c, K')$: then Alice and Bob have the same final session key $\mathtt{SK} = \mathtt{SK}' \leftarrow \mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K')$. If $\mathsf{pw} \neq \mathsf{pw}_A$, we have both inequalities (excepted by chance) $K \neq K' \leftarrow \mathtt{Decaps}(sk, c')$ and $\mathtt{Auth} \neq \mathtt{Auth}'$: then $\mathtt{SK}' \leftarrow \mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K')$ is independent from the $\mathtt{SK}$ computed by Bob, excepted with random equality, which makes no difference from the previous game.

Game $\mathbf{G}_{5.2}$*:* $(c, \mathtt{Auth})$ *comes from* $\mathcal{A}$*.* From the uniqueness of $\mathtt{Auth}$ (from $\mathbf{G}_{0.1}$) in $\Lambda_{\mathtt{H}_1}$, from explicit query to $\mathtt{H}_1$ (or no record at all), $\mathcal{S}$ can extract at most one pair (pw, K) used by $\mathcal{A}$:

- If $\mathsf{pw} = \mathsf{pw}_A$, with $(c, \mathtt{Auth})$: $\mathcal{S}$ computes $K' \leftarrow \mathtt{Decaps}(sk, c)$, $\mathtt{Auth}' \leftarrow \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K')$, and $\mathtt{SK}' \leftarrow \mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}', K')$.
- Else ($\mathsf{pw} \neq \mathsf{pw}_A$, or $\mathsf{pw} = \perp$): $\mathcal{S}$ computes $K' \leftarrow \mathtt{Decaps}(sk, c)$, $\mathtt{Auth}' \leftarrow \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K')$, and aborts.

As we have excluded collisions on $\mathtt{H}_1$, the latter case leads to $\mathtt{Auth} \neq \mathtt{Auth}$, and thus to an abort: this makes no difference from the previous game. As this last game $\mathbf{G}_{5.2}$ is $\mathbf{G}_6$, $|\Pr[\mathbf{G}_6] - \Pr[\mathbf{G}_5]| = 0$.

Game $\mathbf{G_7}$: Random Session Keys. Thanks to the above simulation of the ideal cipher and the random oracle, $\mathcal{S}$ has the ability to extract the tentative password used by the adversary. It will be given access to two boolean functions;

– $\mathtt{GoodPwd}$ with input $(\mathsf{ssid}, P_i, \mathsf{pw})$ that answers whether this is the correct password of party P_i.
– $\mathtt{SamePwd}$ with input $(\mathsf{ssid}, P_i, P_j)$ that answers whether P_i and P_j share the same password.

We first replace session key K generation, for honest players, without knowing the passwords, by using $\mathtt{SamePwd}$ when it did not extract the password and $\mathtt{GoodPwd}$ when it successfully extracted pw, using two private random oracles $\mathtt{H}^*_A$ and $\mathtt{H}^*_K$ onto respectively $\{0,1\}^{\lambda_{\mathtt{H}_1}}$ and $\mathcal{K}$, in $\mathbf{G}_{6.1}$. We then replace the authentication $\mathtt{Auth}$ in $\mathbf{G}_{6.2}$ and lastly the final session key $\mathtt{SK}$ generation in $\mathbf{G}_7 = \mathbf{G}_{6.3}$.

Game $\mathbf{G}_{6.1}$*: Random* K'*.* We first use a private random oracle $\mathtt{H}^*_K$ onto $\mathcal{K}$ to replace K by random K', in some situations:

On Bob's Side: Upon receiving $\mathbf{Epk}$

– From an honest Alice, instead of using K to compute the authentication tag $\mathtt{Auth} \leftarrow \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}_B, K)$ and the final session key $\mathtt{SK} \leftarrow \mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K)$, if $\mathtt{SamePwd}(\mathsf{ssid}, P_i, P_j) = \mathsf{true}$, one sets $K' \leftarrow \mathtt{H}^*_K(\mathsf{ssid}, \mathsf{success})$ otherwise one sets $K' \leftarrow \mathtt{H}^*_K(\mathsf{ssid}, \mathsf{fail}_B)$, and updates the definition of both $\mathtt{Auth} \leftarrow \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}_B, K')$ and $\mathtt{SK} \leftarrow \mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}', K')$.
– From $\mathcal{A}$, with extracted password pw: if $\mathtt{GoodPwd}(\mathsf{ssid}, P_j, \mathsf{pw}) = \mathsf{true}$, one keeps the tag $\mathtt{Auth} \leftarrow \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K)$ and the key $\mathtt{SK} \leftarrow \mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K)$; else one sets $K' \leftarrow \mathtt{H}^*_K(\mathsf{ssid}, \mathsf{fail}_B)$, and updates the definitions: $\mathtt{Auth} \leftarrow \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}_B, K')$ and $\mathtt{SK} \leftarrow \mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}', K')$.

On Alice's Side: Upon receiving $(c, \mathtt{Auth})$

– From an honest Bob, instead of using K to compute the authentication tag $\mathtt{Auth} \leftarrow \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}_A, K)$ and the final session key $\mathtt{SK} \leftarrow \mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K)$, if $\mathtt{SamePwd}(\mathsf{ssid}, P_i, P_j) = \mathsf{true}$, one sets $K' \leftarrow \mathtt{H}^*_K(\mathsf{ssid}, \mathsf{success})$ otherwise one sets $K' \leftarrow \mathtt{H}^*_K(\mathsf{ssid}, \mathsf{fail}_A)$ and lastly updates the definition $\mathtt{Auth}' \leftarrow \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}_A, K')$. In the former case, $\mathtt{Auth} = \mathtt{Auth}'$, since then authentication succeeded $\mathcal{S}$ computes $\mathtt{SK} \leftarrow \mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}', K')$ otherwise $\mathtt{Auth} \neq \mathtt{Auth}'$, $\mathcal{S}$ aborts.
– From $\mathcal{A}$, with extracted password pw: if $\mathtt{GoodPwd}(\mathsf{ssid}, P_i, \mathsf{pw}) = \mathsf{true}$, one keeps the tag $\mathtt{Auth} \leftarrow \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K)$ and the key $\mathtt{SK} \leftarrow \mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, \mathbf{Ec}, K)$; else one sets $K' \leftarrow \mathtt{H}^*_K(\mathsf{ssid}, \mathsf{fail}_A)$, and updates the definitions $\mathtt{Auth}' \leftarrow \mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}_A, K')$, since $\mathtt{Auth} \neq \mathtt{Auth}'$: $\mathcal{S}$ aborts.

Analysis: In this simulation we still use passwords. We replace real keys K by random independent keys K', except when interacting with the adversary that has guessed the password or when an abort occurs due to a failed authentication: $\texttt{Auth} \neq \texttt{Auth}'$. In all the modified sessions, K has been generated from a fresh KEM instance: From an honest Alice, Bob always uses a pk coming from $\texttt{D}$, a (c, K) coming from $\texttt{Encaps}(pk)$ and $\texttt{Auth}$ built honestly; if this comes from $\mathcal{A}$, unless the password was correctly guessed, pk also comes from $\texttt{D}$, (c, K) from $\texttt{Encaps}(pk)$ and $\texttt{Auth}$ built honestly. On Alice's side, unless the password was correctly guessed by the adversary, pk also comes from $\texttt{D}$. However c can be random, but $\mathcal{S}$ uses $\texttt{H}_1$ to extract the password pw and get (sk, pk) from $\texttt{D}$. If at some point either $\mathcal{S}$ can not build $\texttt{Auth}'$ or builds $\texttt{Auth}' \neq \texttt{Auth}$ then it aborts due to a failed authentication. Note that $\mathcal{S}$ does not abort if and only if both (c, K) really come from an $\texttt{Encaps}(pk)$ and $\texttt{Auth}$ has been built honestly. We can thus proceed with a sequence of hybrid games, replacing real keys by random keys.

- On Bob's side, pk comes from $\texttt{D}$, c is simulated by $\mathcal{S}$ and $\texttt{Auth}$ built accordingly, with known (pk, sk) and (c, K), we can simply successively replace (pk, c, K) by (pk, c, K'), using the indistinguishability of the KEM: the gap is bounded by $q'_{\texttt{D}} \cdot \mathsf{Adv}^{\text{ind}}_{\texttt{KEM}}(t)$.
- On Alice's side, the pairs come from $\texttt{D}$ and more importantly $(c, \texttt{Auth})$ has been built honestly otherwise $\mathcal{S}$ would have aborted. Using $\texttt{Auth}$ the simulation is done with known (pk, sk) and (c, K). We can replace (pk, c, K) by (pk, c, K'), using the indistinguishability of the KEM: because the $\texttt{Auth}$ has been honestly computed only one tuple needs to be replaced while $\mathcal{S}$ has knowledge of pw the gap is bounded by $\mathsf{Adv}^{\text{ind}}_{\texttt{KEM}}(t)$.

We thus have $\mid \Pr[\mathbf{G}_{6.1}] - \Pr[\mathbf{G}_6] \mid \leq (1 + q'_{\texttt{D}}) \cdot \mathsf{Adv}^{\text{ind}}_{\texttt{KEM}}(t)$.

Game $\mathbf{G}_{6.2}$*: Random* $\texttt{Auth}$. We now remove knowledge on the passwords from $\mathcal{S}$, it can only use $\texttt{GoodPwd}$ and $\texttt{SamePwd}$. We replace $\texttt{Auth}$ by random $\texttt{Auth}'$ in some situations:

On Bob's Side: Upon receiving **Epk**

- From an honest Alice, instead of using the password to compute the following tag $\texttt{Auth} \leftarrow \texttt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}_B, K)$, one first gets $c \leftarrow\$ \mathcal{C}$ then, if $\texttt{SamePwd}(\mathsf{ssid}, P_i, P_j) = \mathsf{true}$, one updates the generation by $\texttt{Auth} \leftarrow \texttt{H}^*_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{success})$, otherwise one uses the generation $\texttt{Auth} \leftarrow \texttt{H}^*_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{fail})$.
- From $\mathcal{A}$, with extracted password pw: if $\texttt{GoodPwd}(\mathsf{ssid}, P_j, \mathsf{pw}) = \mathsf{true}$, one keeps $\texttt{Auth} \leftarrow \texttt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K)$; else one gets $c \in \mathcal{C}$ and uses the generation $\texttt{Auth} \leftarrow \texttt{H}^*_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{fail})$.

On Alice's Side: Upon receiving $(c, \texttt{Auth})$

- From an honest Bob, instead of using the password to compute the tag $\texttt{Auth}' \leftarrow \texttt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}_A, K)$, if $\texttt{SamePwd}(\mathsf{ssid}, P_i, P_j) = \mathsf{true}$, one updates the generation by $\texttt{Auth}' \leftarrow \texttt{H}^*_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{success})$, otherwise $\mathcal{S}$ aborts because the authentication is bound to fail.

– From $\mathcal{A}$, with extracted password pw: if $\texttt{GoodPwd}(\mathsf{ssid}, P_i, \mathsf{pw}) = \mathsf{true}$, one keeps $\texttt{Auth}' \leftarrow \texttt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K)$; else whether password are different or $\texttt{H}_1$ has not been queried to obtain $\texttt{Auth}$, one aborts as the authentication is bound to fail.

Note that according to $\texttt{H}_2$ behavior, when the $\texttt{Auth}$ is bound to fail (*i.e.* when $\mathsf{status} = \mathsf{fail}$), Bob derives a random key still even though Alice will abort. The environment can make this game fails using two ways. First on Bob's behalf we replaced c built from an $\texttt{Encaps}$ with $c \leftarrow\!\$\ \mathcal{C}$. Since the $\texttt{KEM}$ is anonymous the adversary can distinguish this simulation by querying $q_{\texttt{D}}$ times $\texttt{D}$ to break the anonimity. Lastly it can try to query $\texttt{H}_1$ on an input that has been replaced by $\texttt{H}_1^*$. But in the previous game, all the K that are in such changes are truly random, and there are at most $2q_s$ such changes, where q_s is the number of sessions: We thus have $\mid \Pr[\mathbf{G}_{6.2}] - \Pr[\mathbf{G}_{6.1}] \mid \leq q_{\texttt{D}} \cdot \mathsf{Adv}^{\mathrm{ano}}_{\texttt{KEM}}(\mathcal{A}) + q_{\texttt{H}_1} \cdot q_s / 2^{\lambda_k}$, where λ_k is the length of K.

Game $\mathbf{G}_{6.3}$*: Random* $\texttt{SK}$*.* We now replace $\texttt{SK}$ by random $\texttt{SK}'$, in some situations:

On Bob's Side: Upon receiving **Epk**

– From an honest Alice, instead of using K to compute the key $\texttt{SK} \leftarrow \texttt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \texttt{Auth}, K)$, if $\texttt{SamePwd}(\mathsf{ssid}, P_i, P_j) = \mathsf{true}$, one updates the generation by $\texttt{SK} \leftarrow \texttt{H}_2^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \texttt{Auth}, \mathsf{success})$, otherwise one uses the generation $\texttt{SK} \leftarrow \texttt{H}_2^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \texttt{Auth}, \mathsf{fail})$.

– From $\mathcal{A}$, with extracted password pw: If $\texttt{GoodPwd}(\mathsf{ssid}, P_j, \mathsf{pw}) = \mathsf{true}$, one keeps $\texttt{SK} \leftarrow \texttt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \texttt{Auth}, K)$; else one uses the generation $\texttt{SK} \leftarrow \texttt{H}_2^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \texttt{Auth}, \mathsf{fail})$.

On Alice's Side: Upon receiving $(c, \texttt{Auth})$

– From an honest Bob, instead of using K to compute the key $\texttt{SK} \leftarrow \texttt{H}_2(\mathsf{ssid}, P_i, P_j, \mathsf{pw}, \mathbf{Epk}, c, K)$, if $\texttt{SamePwd}(\mathsf{ssid}, P_i, P_j) = \mathsf{true}$, one updates the generation by $\texttt{SK} \leftarrow \texttt{H}_2^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \texttt{Auth}, \mathsf{success})$.

– From $\mathcal{A}$, with extracted password pw: if $\texttt{GoodPwd}(\mathsf{ssid}, P_i, \mathsf{pw}) = \mathsf{true}$ and $\texttt{Auth} = \texttt{Auth}'$ therefore one keeps $\texttt{SK} \leftarrow \texttt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \texttt{Auth}', K')$.

The only way for the environment to detect the difference is to have a call $\texttt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \texttt{Auth}, K)$ that has been replaced by a call to $\texttt{H}_2^*$. But in the previous game, all the K that are in such changes are truly random, and there are at most $2q_s$ such changes, where q_s is the number of sessions: We thus have $\mid \Pr[\mathbf{G}_{6.3}] - \Pr[\mathbf{G}_{6.2}] \mid \leq q_{\texttt{H}_2} \cdot q_s / 2^{\lambda_k}$, where λ_k is the length of K.

As this last game $\mathbf{G}_{6.3}$ is $\mathbf{G}_7$, $\mid \Pr[\mathbf{G}_7] - \Pr[\mathbf{G}_6] \mid \leq (1 + q'_{\texttt{D}}) \cdot \mathsf{Adv}^{\mathrm{ind}}_{\texttt{KEM}}(t) + q_{\texttt{D}} \cdot \mathsf{Adv}^{\mathrm{ano}}_{\texttt{KEM}}(\mathcal{A}) + (q_{\texttt{H}_1} + q_{\texttt{H}_2}) \cdot q_s / 2^{\lambda_k}$.

Game $\mathbf{G_8}$: Adding the Full $\mathcal{F}_{pwKE\text{-}sA}$ Interface. In this game we add the full $\mathcal{F}_{pwKE\text{-}sA}$ interface to fully model the ideal world. First, $\mathcal{S}$ simulates its use of $\texttt{GoodPwd}$ by querying $\texttt{TestPwd}$ to $\mathcal{F}_{pwKE\text{-}sA}$ on input $(\mathsf{ssid}, P_i, \mathsf{pw})$ to test if pw is the password associated to P_i in ssid. Secondly, $\texttt{SamePwd}$ on $(ssid, P_i, P_j)$ perfectly embodies the equality that $\mathcal{F}_{pwKE\text{-}sA}$ does internally with knowledge of the passwords when using records $(P_i, P_j, \mathsf{pw}, \mathsf{status})$. Note that here status represents client and server unline the previous game with success and fail. Then, for key generation, when a value K can be computed, and $\mathtt{SK} \leftarrow \mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K)$, then one queries the ideal functionnality $\mathcal{F}_{pwKE\text{-}sA}$ on $(\mathtt{NewKey}, \mathsf{ssid}, P, \mathtt{SK})$, for any party P.

Let us show this provides the same output as in the previous game: First, in the simulation, if $\mathcal{S}$ has extracted the password used by the adversary, then a TestPwd query has been sent to $\mathcal{F}_{pwKE\text{-}sA}$. According to the ideal functionality two cases arise, according to the correct guess:

– If the guess is incorrect: the record is marked as interrupted.
– If the guess is correct: the record is marked as compromised.

If the session is compromised, $\mathcal{F}_{pwKE\text{-}sA}$ returns $(ssid, \mathtt{SK})$ to P_i on query to NewKey with $(\mathsf{ssid}, P_i, \mathtt{SK})$. These are the cases where we kept the definition of SK with $\mathtt{H}_2$.
If the session is interrupted it returns a random $\mathtt{SK}'$ to the server and set an error for the client, which are the cases where we used $\mathtt{H}_1^*$ with fail. This behavior of NewKey is exactly the one from that has been simulated by $\mathcal{S}$ and therefore remains indistinguishable.

Now assuming that $\mathcal{S}$ could not extract a password: it has sent random flow **Epk** and $(c, \mathtt{Auth})$ and used its private oracles $\mathtt{H}_1^*$ and $\mathtt{H}_2^*$ to build a random session key for each party. Since $\mathcal{S}$ does not know the parties' passwords it used SamePwd to obtain know if they use the same password. By definition of the NewKey interface:

– Two honest parties in a fresh session using the same password, derive the same session key, because of a success status.
– Two honest parties in a fresh session not using the same passwords: the client is returned abort from $\mathcal{F}_{pwKE\text{-}sA}$ while the server obtains a random session key because of the fail status.

Hence, we have $\mid \Pr[\mathbf{G}_8] - \Pr[\mathbf{G}_7] \mid = 0$, and this game is perfectly indistinguishable from the ideal world.

The global gap is thus:

$$\begin{aligned}\mid \Pr[\mathbf{G}_9] - \Pr[\mathbf{G}_0] \mid \leq\ & q'_{\mathtt{D}} \cdot \mathsf{Adv}^{\text{fuzzy}}_{\mathtt{KEM}}(t) + (1 + q'_{\mathtt{D}}) \cdot \mathsf{Adv}^{\text{ind}}_{\mathtt{KEM}}(t) + q_{\mathtt{D}} \cdot \mathsf{Adv}^{\text{ano}}_{\mathtt{KEM}}(\mathcal{A}) \\ & + {q'_{\mathtt{H}_1}}^2 \cdot 2^{-\lambda_{\mathtt{H}_1}-1} + q_{\mathtt{E}}^2 \cdot 2^{-\lambda_p - 1} + q_s \cdot 2^{-\lambda_{\mathtt{H}_1}} \\ & + (q_{\mathtt{H}_1} + q_{\mathtt{H}_2}) \cdot q_s \cdot 2^{-\lambda_k}\end{aligned}$$

where we denote the global numbers of queries asked by the adversary $q_{\mathtt{E}}$, $q_{\mathtt{D}}$, and $q_{\mathtt{H}}$. But we also need to count the number of queries to $\mathtt{H}_1$ and $\mathtt{D}$ by the adversary and the simulator, at some point of the simulation:

– the global number of queries asked to $\mathtt{D}$ is q_s (for initialization by Alice or answer by Bob) plus $q_{\mathtt{D}}$;
– the global number of queries asked to $\mathtt{H}_1$ is q_s (for answer by Bob) plus $q_{\mathtt{H}}$.

Laslty on `NewSession`, $\mathcal{S}$ sends **Epk** on behalf of the client and does nothing for the server Only $\mathcal{F}_{pwKE}$ does his internal computation.
The simulation of both $\mathtt{H}_1$ and $\mathtt{H}_2$ are independant of the status component as it is handle by $\mathcal{F}_{pwKE\text{-}sA}$. Additionally we have,

$$\begin{aligned} |\Pr[\mathbf{G}_9] - \Pr[\mathbf{G}_0]| \leq\ & (q_s + q_{\mathtt{D}}) \cdot \mathsf{Adv}^{\text{fuzzy}}_{\mathtt{KEM}}(t) + (q_s + q_{\mathtt{D}} + 1) \cdot \mathsf{Adv}^{\text{ind}}_{\mathtt{KEM}}(t) \\ & + q_{\mathtt{D}} \cdot \mathsf{Adv}^{\text{ano}}_{\mathtt{KEM}}(\mathcal{A}) + (q_{\mathtt{H}_1} + q_s)^2 \cdot 2^{-\lambda_{\mathtt{H}_1} - 1} \\ & + (q_{\mathtt{H}_1} + q_{\mathtt{H}_2}) \cdot q_s \cdot 2^{-\lambda_k} + q_{\mathtt{E}}^2 \cdot 2^{-\lambda_p - 1} + q_s \cdot 2^{-\lambda_{\mathtt{H}_1}} \end{aligned}$$

```
On $\mathtt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K)$
If $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K, \mathtt{Auth}) \in \Lambda_{\mathtt{H}_1}$
    return $\mathtt{Auth}$
Else:
    sample $\mathtt{Auth} \leftarrow\$ \{0,1\}^{\lambda_{\mathtt{H}_1}}$ and record
    $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K, \mathtt{Auth}) \in \Lambda_{\mathtt{H}_1}$
    return $\mathtt{Auth}$
```

```
On $\mathtt{H}_1^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw})$
If $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, \mathtt{Auth}) \in \Lambda_{\mathtt{H}_1^*}$
    return $\mathtt{Auth}$
Else:
    sample $\mathtt{Auth} \leftarrow\$ \{0,1\}^{\lambda_{\mathtt{H}_1}}$ and record:
    $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, \mathtt{Auth}) \in \Lambda_{\mathtt{H}_1^*}$
    return $\mathtt{Auth}$
```

```
On $\mathtt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K)$
If $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K, \mathtt{SK}) \in \Lambda_{\mathtt{H}_2}$
    return $\mathtt{SK}$
Else:
    sample $\mathtt{SK} \leftarrow\$ \{0,1\}^{\lambda_{\mathtt{H}_2}}$ and record
    $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, K, \mathtt{SK}) \in \Lambda_{\mathtt{H}_2}$
    return $\mathtt{SK}$
```

```
On $\mathtt{H}_2^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth})$
If $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, \mathtt{SK}) \in \Lambda_{\mathtt{H}_2^*}$
    return $\mathtt{SK}$
Else:
    sample $\mathtt{SK} \leftarrow\$ \{0,1\}^{\lambda_{\mathtt{H}_2}}$ and record:
    $(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathtt{Auth}, \mathtt{SK}) \in \Lambda_{\mathtt{H}_2^*}$
    return $\mathtt{SK}$
```

```
On $\mathtt{E}(\mathsf{ssid}||\mathsf{pw}, pk)$
If $\exists(\mathsf{ssid}, \mathsf{pw}, pk, \star, \star, \mathbf{Epk}) \in \Lambda_{\mathtt{E}}$ :
    returns $\mathbf{Epk}$.
Otherwise: $\mathbf{Epk} \leftarrow\$ \mathcal{P}'$.
If: $\exists(\star, \star, \star, \star, \star, \mathbf{Epk}) \in \Lambda_{\mathtt{E}}$ :
    $\mathcal{S}$ aborts.
Else: $\mathcal{S}$ records :
    $(\mathsf{ssid}, \mathsf{pw}, pk, \perp, \mathtt{E}, \mathbf{Epk}) \in \Lambda_{\mathtt{E}}$
    returns $\mathbf{Epk}$.
```

```
On $\mathtt{D}(\mathsf{ssid}||\mathsf{pw}, \mathbf{Epk})$
If $\exists(\mathsf{ssid}, \mathsf{pw}, pk, \star, \star, \mathbf{Epk}) \in \Lambda_{\mathtt{E}}$: returns $pk$.
Else, $\mathcal{S}$ builds $(pk, sk) \leftarrow \mathtt{KeyGen}(1^\kappa)$
records $(\mathsf{ssid}, \mathsf{pw}, pk, sk, \mathtt{D}, \mathbf{Epk}) \in \Lambda_{\mathtt{E}}$
returns $pk$.
```

Fig. 10. Simulation of the ideal ciphers and the random oracle in proof of Theorem 2.

```
On $(\texttt{NewSession}, \mathsf{ssid}, \mathsf{role}, P_i, P_j)$ from $\mathcal{F}_{pwKE\text{-}sA}$

If $P_i = \mathsf{client}$ :
    $\mathbf{Epk} \leftarrow\!\$\, \mathcal{P}'$, sends $\mathbf{Epk}$

Upon $\mathsf{server}$ $P_i$ receiving $\mathbf{Epk}$

If $\exists(\mathsf{ssid}, \mathsf{pw}, pk, \star, \star, \mathbf{Epk})$:
    Upon positive answer after querying: $(\texttt{TestPwd}, \mathsf{ssid}, P_i, \mathsf{pw})$ to $\mathcal{F}_{pwKE\text{-}sA}$ :
        compute: $(c, K) \leftarrow \texttt{Encaps}(pk)$,
        $\texttt{Auth} \leftarrow \texttt{H}_1(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K, \texttt{Auth})$,
        $\texttt{SK} \leftarrow \texttt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \texttt{Auth}, K, \texttt{SK})$
Else:
    sample: $\mathbf{Ec} \leftarrow\!\$\, \mathcal{C}'$,
    $\texttt{Auth} \leftarrow \texttt{H}_1^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c)$
    $\texttt{SK} \leftarrow \texttt{H}_2^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c)$
send: $(\mathbf{Ec}, \texttt{Auth})$ to $P_j$ and $(\texttt{NewKey}, \mathsf{ssid}, P_i, \texttt{SK})$ to $\mathcal{F}_{pwKE\text{-}sA}$

Upon $\mathsf{client}$ $P_i$ receiving $(c, \texttt{Auth})$

If $\exists(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \mathsf{pw}, K, \texttt{Auth}) \in \Lambda_{\texttt{H}_1}$:
    Upon positive answer after querying: $(\texttt{TestPwd}, \mathsf{ssid}, P_i, \mathsf{pw})$ to $\mathcal{F}_{pwKE\text{-}sA}$ :
        get: $sk, pk \leftarrow \texttt{D}_1(\mathsf{ssid}||\mathsf{pw}, \mathbf{Epk})$
        extract $K$ or compute: $K' \leftarrow \texttt{Decaps}(sk, c)$
        If $K = K'$ : get $\texttt{SK} \leftarrow \texttt{H}_2(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \texttt{Auth})$
        Else $\mathsf{abort}$
Else:
    If it comes from $\mathcal{A}$, $\mathsf{abort}$
    If it comes from $P_i$ :
    get: $\texttt{Auth} \leftarrow \texttt{H}_1^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c)$
    get: $\texttt{SK} \leftarrow \texttt{H}_2^*(\mathsf{ssid}, P_i, P_j, \mathbf{Epk}, c, \texttt{Auth})$
send: $(\texttt{NewKey}, \mathsf{ssid}, P_i, \texttt{SK})$ to $\mathcal{F}_{pwKE\text{-}sA}$
(Note that $\texttt{SK} = \mathsf{error}$ is possible according to $\mathcal{F}_{pwKE\text{-}sA}$)
```

Fig. 11. Simulation of the protocol from the client and server point of view in proof of Theorem 2.

C IF

Functionnality $\mathcal{F}_{RO}$

On security parameter k, with parties $P_1, \ldots, P_n$ and adversary $\mathcal{S}$.

1. $\mathcal{F}_{RO}$ keeps a list L (which is initially empty) of pairs of bitstrings.
2. Upon receiving a value (sid, m) (with $m \in \{0,1\}^*$) from some party P_i or from $\mathcal{S}$, do:
   - If there is a pair $(m, \bar{h})$ or some $(\bar{h} \in \{0,1\}^k)$ in the list L, set $h := \bar{h}$.
   - If there is no such pair, choose uniformly $h \in \{0,1\}^k$ and store the pair (m, h) in L.

Fig. 12. $\mathcal{F}_{RO}$: the ideal Functionality of the random oracle.

Functionnality $\mathcal{F}_{IC}$

On security parameter k, interact with an adversary $\mathcal{S}$ and with a set of dummy parties $P_1, \ldots, P_n$:

- $\mathcal{F}_{IC}$ keeps a (initially empty) list L containing 3-tuples of bitstrings and a number of initially empty sets $C_{key,sid}$, $M_{key,sid}$.
- **Upon receiving a query (sid, ENC, key, m) (with $m \in \{0,1\}^k$) from some party P_i or $\mathcal{S}$, do:**
  - If there is a 3-tuple $(key, m, \bar{c})$ for some $\bar{c} \in \{0,1\}^k$ in the list L, set $c := \bar{c}$.
  - If there is no such record, choose unformly $c \in \{0,1\}^k \backslash C_{key,sid}$

  Once c is set, replu to the activating machine with (sid, c)
- **Upon receiving a query (sid, DEC, key, c) (with $c \in \{0,1\}^k$) from party P_i or $\mathcal{S}$, do:**
  - If there is a 3-tuple $(key, \bar{m}, c)$ for some $\bar{m} \in \{0,1\}^k$ in L, set $m := \bar{m}$
  - If there is no such record, choose uniformly $m \in \{0,1\}^k \backslash M_{key,sid}$ which is the set consisting of plaintexts not already used with key and sid. Next, it stores the 3-tuple $(key, m, c) \in L$ and set $M_{key,sid} \leftarrow M_{key,sid} \bigcup \{m\}$.

  Once m is set, reply to the activating machine with (sid, m).

Fig. 13. $\mathcal{F}_{IC}$: the ideal Functionality of an ideal cipher.

Adaptive Corruption

On $(\texttt{AdaptiveCorruption}, sid, P_i)$ from $\mathcal{A}$, if there exists a record $< sid, P_i, P_j, pw >$ then :

- if (sid, K) was output to P_i, send (sid, P_j, pw, K) to $\mathcal{A}$.
- otherwise send $(sid, P_j, pw, \perp)$ to $\mathcal{A}$.

Fig. 14. Adaptive Corruption enforced by the environment $\mathcal{Z}$.