Towards post-quantum secure PAKE - A tight security proof for OCAKE in the BPR model

Nouri Alnahawi[1] [*], Kathrin Hövelmanns[2], Andreas Hülsing[2] [**], Silvia Ritsch[2] [***], and Alexander Wiesmaier[1]

[1] Darmstadt University of Applied Sciences, Germany
[2] Eindhoven University of Technology, The Netherlands

Abstract. We revisit OCAKE (ACNS 23), a generic recipe that constructs password-based authenticated key exchange (PAKE) from key encapsulation mechanisms (KEMs) in a black-box way. This allows to potentially achieve post-quantum security by instantiating the KEM with a post-quantum KEM like KYBER. It was left as an open problem to further adapt the proof such that it also holds against quantum attackers. The security proof is given in the universal composability (UC) framework, which is commonly used to model and prove security of PAKE. So far, however, it is not known how to model or prove computational UC security against quantum adversaries. Even more so, if the proof makes use of idealized primitives like random oracles or ideal ciphers.

To pave the way towards reasoning post-quantum security, we therefore resort to a (still classical) game-based security proof in the BPR model (EUROCRYPT 2000). We consider this a crucial stepping stone towards a full proof of post-quantum security.

We prove security of (a minor variation of) OCAKE generically, assuming the underlying KEM satisfies common notions of ciphertext indistinguishability, anonymity, and (computational) public key uniformity. To achieve tight security bounds, we relate security of OCAKE to multi-user variants of the aforementioned properties.

We provide a full detailed proof – something often omitted in publications concerned with game-based security of PAKE. As a side-contribution, we demonstrate how to handle password guesses in a game-based proof in detail. Something we were unable to find in the existing literature.

Keywords: Public-key cryptography, password-based authenticated key exchange, PAKE, CAKE, OCAKE, post-quantum cryptography, ROM, game-based security.

1 Introduction

A central problem of secure communication is how to securely agree on a shared secret key via public communication. The generic solution is called an authenticated key exchange (AKE) protocol, in which commonly public key cryptography is used to agree on the shared secret. This is the basis of most modern secure communication protocols, including TLS, SSH, or WireGuard. The drawback of this solution is that it requires users to hold a cryptographic key pair for authentication. As cryptographic keys are hard to memorize, they require secure storage with all the related challenges for usability. Hence, in many scenarios only the server is authenticated during the AKE protocol. Users are often authenticated via the use of passwords within the already

[*] N.A. was supported by National Research Center for Applied Cyber-Security ATHENE.
[**] A.H. was supported by an NWO VIDI grant (Project No. VI.Vidi.193.066).
[***] S.R. is part of the Quantum-Safe Internet (QSI) ITN which received funding from the European Union's Horizon-Europe programme as Marie Sklodowska-Curie Action (PROJECT 101072637 - HORIZON -MSCA-2021-DN-01)

established communication which is secured via the shared secret. This is the case as passwords are far easier to handle by humans. However, this means that afterwards additional measures have to be taken to link the authentication to the session secured via the shared secret. A way out of this is to use a password for the purpose of authentication in an AKE. Such protocols are called password-authenticated key exchange (PAKE). In general, PAKE allow the use of any low-entropy shared secret (like a password or a PIN) to provide the agreement with authentication. Hao and van Oorschot classify in their SoK on PAKE[HvO22] real-world use-cases of PAKE protocols and the currently used PAKEs related to them. These include credential recovery using the SRP-6a protocol in iCloud; device pairing (mostly IoT or embedded devices), using the PACE protocol in eIDs or eMRTDs to prevent skimming, as well as Dragonfly in WPA3 (standard for WiFi connection establishment); and E2E secure channel establishment using the J-PAKE protocol in Thread. A few years ago, interest in the design and theory surrounding PAKE was increased further when the Crypto Forum Research Group (CFRG) - advisory body to the Internet Engineering Task Force (IETF) - performed a selection process for new PAKE standards. The defined requirements[3] emphasized high efficiency and simultaneously high security, supported by a formal security proof.

The quantum threat. While the CFRG announced two winners in 2020, all proposals (including the winners OPAQUE [JKX18] and CPace [AHH23]) have in common that they rely on the computational hardness of the Diffie-Hellman (DH) problem – something they share with most currently deployed public-key cryptography. Since Shor famously showed how to solve this problem on a quantum computer, public-key cryptography based on DH – including the proposed PAKE protocols – do not offer resilience against quantum attacks.

As a first step towards dealing with the 'quantum threat', the National Institute of Standards and Technology (NIST) posed a call for proposals in 2017 with the goal to develop quantum-resistant standards for public-key encryption (PKE) and digital signature schemes, which are the most fundamental building blocks underpinning public-key cryptography. More accurately, rather than aiming at PKE schemes, NIST aimed at key encapsulation mechanisms (KEMs). A KEM is similar to a PKE but focused on the use-case of establishing a shared secret by sending key in encrypted form. This allows the encapsulation algorithm to internally chose the key, instead of taking it as an input, and returning the key together with a ciphertext that "encapsulates" it. This change in functionality allows for more efficient constructions as the key cannot be adversarially chosen during attacks. The NIST process recently selected Kyber [BDK$^+$18] as KEM and Dilithium [DKL$^+$18], Falcon [PFH$^+$22], and SPHINCS$^+$ [BHK$^+$19] as signatures for standardization. In the context of this work we are only interested in KEMs. It should be noted that KEMs are fundamentally different from the Diffie-Hellman key exchange (DHKX), although they serve the same purpose. The DHKX is a non-interactive key exchange with a lot of additional algebraic structure. In comparison, when KEMs are used for key exchange, the resulting protocol is interactive, and they do not provide additional structure generically (although specific proposals do). While NIST is continuing the selection process for further KEM and signature schemes, there is no process for NIKE. The reason is the lack of an efficient candidate with reliable security at this time (first proposals exist though [CLM$^+$18, DKS18, RS06, Cou06]).

Designing post-quantum PAKE. A major challenge regarding the transition to post-quantum secure systems, is to transform existing protocols into post-quantum secure protocols replacing quantum-vulnerable building blocks by the available KEM and signatures. The general challenge that also concerns PAKE is that the NIST proposals cannot replace the Diffie-Hellman key exchange in PAKE protocols in a 'plug-n-play' way: most PAKE protocols rely on the additional

[3] Specified in datatracker.ietf.org/doc/html/rfc8125, and expanded upon in ietf.org/proceedings/104/slides/slides-104-cfrg-pake-selection-01.pdf

algebraic properties of the group operation in DHKX which are not known to be offered by KEMs. This gave rise to the requirement to design new PAKE protocols, preferably in a way that

- is versatile, i.e., a way that works for various PQC proposals or even pre- and post-quantum hybrids (instead of using the internal workings of a specific proposal);
- works with the proposed algorithms (rather than making it necessary to introduce new primitives);
- avoids complex mapping operations used, e.g., by elliptic-curve-based protocols;
- and satisfies state-of-the-art security notions (supported by a formal security proof).

The first property is motivated by the idea of crypto agility, i.e., the option to easily replace a building block in case of successful cryptanalysis. However, it also allows to offer the possibility for selecting different candidates depending on specific performance requirements like, e.g., fast computations or low memory consumption.

A candidate proposal that aims at fulfilling the above requirements is OCAKE, recently proposed by Beguinet, Chevalier, Pointcheval, Ricosset, and Rossi [BCP$^+$23]. OCAKE is based on the EKE paradigm [BM92], but replaces the need for Diffie-Hellman by building generically on suitable KEMs, thereby setting a foundation to build quantum-resistant PAKE.

Towards post-quantum security of PAKEs. Modern security notions and proofs for PAKE are usually given in the universal composability (UC) framework introduced in [Can01] (see, e.g., [CHK$^+$05, BBC$^+$13, Sho20, AHH21, ABR$^+$21, BCP$^+$23]). This is also the case for OCAKE. While security proofs in the UC framework are desirable in the sense that UC-proven building blocks can always be composed securely, they come with a limitation when addressing post-quantum security: so far, we are not aware of works that consider computational security against quantum attackers in the UC framework. Hence, it is not known how these proofs can be translated into a setting considering quantum adversaries.

At the same time, there is continuous progress in lifting game-based security results to a setting with quantum adversaries. Indeed, up to minor complications, such lifts are straight-forward as long as no idealized models are used [Son14]. However, when proofs are given in idealized models like the random oracle model (ROM) and/or the ideal cipher (IC) model lifting is less straight-forward (and this is the case for PAKE). Both the ROM and the IC model do not account for quantum attacks and therefore make it necessary to adapt the models and the proofs. The quantum-accessible ROM (QROM) [BDF$^+$11] is somewhat well understood by now. Ongoing efforts to develop the necessary techniques for lifting proofs to the QROM together are well under way (see, e.g., [Zha19, DFMS21, CFHL21, GHHM21, HHM22, DFMS22]). Similar results for the quantum-accessible ideal cipher model are still extremely limited but a model exists and first proofs have been done [HY18].

This suggests that game-based security notions and proofs may be a good target for proving security against quantum attacks. There are several game-based security models for PAKE [BPR00, AFP05, Lan16]. However, detailed formal proofs for PAKE protocols that could be lifted are lacking.

Our contribution. In this work, we progress towards a PAKE protocol with proven security against quantum adversaries. Towards this end we revisit the security of a minor variation of OCAKE. We present a rigorous game-based security proof of the protocol in the BPR model for PAKE proposed by Bellare, Pointcheval, and Rogaway [BPR00]. We give a concrete security bound rather than an asymptotic relation, thereby allowing to reason about concrete parameter instantiations. To achieve a tight bound, we make use of multi-user security notions.

As a side contribution, we show how to formally treat password guesses in a detailed game-based proof. So far, we are only aware of detailed proofs which hide this step in a proof in the generic group model[BFK09]. Interestingly, in UC, this step is easy to formalize, but verifying the security reasoning can be challenging.

Our proposal differs from OCAKE in two minor points. First, we omit session identifiers which are included in OCAKE to enable a proof in the UC framework but not necessary for a game-based proof. Second, we consciously add a final key confirmation message that achieve explicit mutual authentication. This is not necessary to prove security in the BPR model but we consider explicit authentication a relevant feature of a protocol (and BPR already discuss that it can be added by adding key confirmation).

A limitation. Although we ultimately aim for security against quantum adversaries, our proof is still in the (classical) ideal cipher and random oracle model. While it would have likely been possible to replace the ROM by the QROM for this proof, handling the ideal cipher seems more challenging due to the limited known proof-techniques. This work presents a solid foundation to start future in work in which either new techniques to lift results with ideal ciphers are developed, or the use of the ideal cipher is omitted.

Organisation of this paper. After recalling basic notions including the relevant security notions for KEMs in Section 2, we describe the CAKE protocol in Section 3. We recall the BPR security model for PAKE in Section 4, and then prove CAKE secure in Section 5.

Acknowledgements. We would like to thank Thomas Pöppelmann for valuable discussions about the design of PAKE protocols, and Afonso Arriaga, Manuel Barbosa, Stanislaw Jarecki, and Marjan Skrobot for valuable discussions about their security proofs.

2 Preliminaries

2.1 The Ideal Cipher Model

We prove security of the PAKEM protocol in the ideal cipher model [Sha49],[Bla06]. Analogously to the random oracle model (RO) for hash functions, the ideal cipher (IC) is an idealized description of a block cipher.

Definition 1 (Block Cipher (BC)). *A block cipher of block length n and with key length k consists of two algorithms $\mathtt{BC.enc} : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and $\mathtt{BC.dec} : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ where it holds that for every plaintext $m \in \{0,1\}^n$ and key $k \in \{0,1\}^k$, decryption undoes encryption: $\mathtt{IC.dec}(k, \mathtt{IC.enc}(k,m)) = m$.*

Definition 2 (Ideal Cipher (IC)). *An ideal cipher is a collection of random permutations indexed by a key, to which all parties (including the adversary) are given oracle access. I.e., it is a pair of random functions $\mathtt{IC.enc}, \mathtt{IC.dec} : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$, such that $\mathtt{IC.dec}(k, \mathtt{IC.enc}(k,m)) = m$ and $\mathtt{IC.enc}(k, \mathtt{IC.dec}(k,m)) = m$ for all k, m in $\mathcal{K} \times \mathcal{M}$.*

2.2 Key Encapsulation Mechanisms

Definition 3 (Key Encapsulation Mechanisms (KEMs)). *A KEM is a triple of algorithms $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap})$, together with a public key space $\mathcal{PK}$ and secret key space $\mathcal{SK}$.*

- $\mathsf{KGen} \to (pk, sk)$*: On empty input* **return** *key pair (pk, sk), where pk also defines a finite key space $\mathcal{K}$ and a ciphertext space $\mathcal{C}$.*
- $\mathsf{Encap}\ (pk) \to (c, k)$*: On input pk* **return** *a tuplet $(K, c) \in \mathcal{K} \times \mathcal{C}$. We call c the* encapsulation *of the key K.*
- $\mathsf{Decap}\ (sk, c) \to k$*: On input sk and ciphertext c deterministically* **return** *a key $K \in \mathcal{K}$.*

Definition 4 ($\delta-$Correctness (average-case)). *We say that* KEM *is average-case* $(1-\delta)$-*correct if*

$$\Pr[\mathsf{Decap}(sk, c) = K | (c, K) \leftarrow\$ \mathsf{Encap}(pk)] \geq 1 - \delta,$$

where the probability is taken over $(pk, sk) \leftarrow \mathsf{KGen}()$ *and the random coins of* Encap.

Security of KEM We define two security notions for KEMs, ANOnymity under Chosen Plaintext Attacks (ANO-CPA) [BBDP01], [GMP22] and INDindistinguishability under chosen-plaintext attacks (IND-CPA), through their respective security experiments in figure 1. For adversaries $\mathcal{A}$ and $\mathcal{A}$', we define the advantages as:

$$\mathbf{Adv}^{\mathsf{ANO\text{-}CPA}}_{\mathsf{KEM}}(\mathcal{A}) := |\Pr[\mathsf{ANO\text{-}CPA}^{b=0}(\mathcal{A})] - \Pr[\mathsf{ANO\text{-}CPA}^{b=1}(\mathcal{A})]| \text{ and}$$
$$\mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{KEM}}(\mathcal{A}') := |\Pr[\mathsf{IND\text{-}CPA}^{b=0}(\mathcal{A}')] - \Pr[\mathsf{IND\text{-}CPA}^{b=1}(\mathcal{A}')]|$$

```
ANO-CPA^{b=0}(A)                                   | ANO-CPA^{b=1}(A)
$(pk_0, sk_0) \leftarrow\$ \mathsf{KGen}$
$(pk_1, sk_1) \leftarrow\$ \mathsf{KGen}$
$(c_0^*, K_0^*) \leftarrow\$ \mathsf{Encap}(pk_0)$  | $(c_1^*, K_1^*) \leftarrow\$ \mathsf{Encap}(pk_1)$
$b' \leftarrow \mathcal{A}(pk_0, pk_1, (c_0^*, K_0^*))$ | $b' \leftarrow \mathcal{A}(pk_0, pk_1, (c_1^*, K_1^*))$
return $b = b'$
```

(a)

```
IND-CPA^{b=0}(A)                      | IND-CPA^{b=1}(A)
$(pk, sk) \leftarrow\$ \mathsf{KGen}$
$(c^*, K^*) \leftarrow\$ \mathsf{Encap}(pk)$
                                      | $K_\$^* \xleftarrow{unif} \mathcal{K}$
$b' \leftarrow \mathcal{A}(pk, c^*, K^*)$ | $b' \leftarrow \mathcal{A}(pk, c^*, K_\$^*)$
return $b = b'$
```

(b)

```
PKU^{b=0}(A)                          | PKU^{b=1}(A)
$(pk, sk) \leftarrow\$ \mathsf{KGen}$ | $pk_\$ \leftarrow\$ \mathcal{PK}$
$b' \leftarrow \mathcal{A}(pk)$       | $b' \leftarrow \mathcal{A}(pk_\$)$
return $b = b'$
```

(c)

Fig. 1: The CPA security games for KEM: anonymity (a), indistinguishability (b) and public-key uniformity (c).

Uniformity of Public Keys We will use in our security proof that any element of the public key space can sufficiently well be *explained as* being a public key generated by the key generation algorithm. Concretely, we formalize this below via a public-key uniformity game that asks the attacker to distinguish honestly generated public keys from ones chosen uniformly at random in the key space. This computational property was also described as *fuzziness* [BCP+23] where it was also proven for the post-quantum KEM CHRYSTALS-Kyber. There have also been statistical definitions, for example Bradley et al [BCJ+19] show that for discrete-log based PKE schemes, public keys are uniformly chosen group elements.

Definition 5. *We say a* KEM *satisfies the public-key uniformity property if its key generation algorithm outputs public keys that are (computationally) indistinguishable from random elements in the public-key space:*

$$\mathbf{Adv}_{\mathsf{KEM}}^{\mathsf{PKU}}(\mathcal{A}) := |\Pr[\mathsf{PKU}^{b=0}(\mathcal{A})] - \Pr[\mathsf{PKU}^{b=1}(\mathcal{A})]|, \textit{ and}$$

where PKU *is shown in figure 1c.*

Multi-user security notions. We define multi-user (and multi-challenge) security notions that model settings where an adversary is given a multiple challenges associated with different key pairs and *wins* if there is a successful attack on some security property for *any* of these.[4] Multi-user security notions have been introduced in [BBM00], including a generic reduction between multi-user and single-user indistinguishability ($(n, q_C) - \mathsf{IND\text{-}CPA}$ below) for public-key encryption. A multi-instance counterpart for key encapsulation mechanisms was given in [GKP18]. We will also use multi-user notions for anonymity ($()$-ANO -CPA) and public-key uniformity (-PKU): For n key pairs and a maximum number of challenge queries q_C, the $(n, q_C)-$ $\mathsf{IND\text{-}CPA}$ experiment is shown in figure 2, and the $(n, q_C) - \mathsf{ANO\text{-}CPA}$ experiment in figure 3. [DHK+21]

```
**Exp** $(n, q_C) - \mathsf{IND\text{-}CPA}^{b=0}(\mathcal{A})$    $(n, q_C) - \mathsf{IND\text{-}CPA}^{b=1}$
for $j \in [n]$
    $(pk_j, sk_j) \leftarrow\!\!\$\ \mathsf{KGen}$
$\boldsymbol{pk} \leftarrow (pk_1, \ldots, pk_n)$
$b' \leftarrow \mathcal{A}^{\mathtt{Chall}_{q_C}^{b=0}(.)}(\boldsymbol{pk})$    $b' \leftarrow \mathcal{A}^{\mathtt{Chall}_{q_C}^{b=1}(.)}(\boldsymbol{pk})$
**return** $b = b'$
```

```
Oracle $\mathtt{Chall}_{q_C}^{b=0}(j)$    $\mathtt{Chall}_{q_C}^{b=1}(j)$
$(c, K) \leftarrow\!\!\$\ \mathsf{Encap}(pk_j)$    $K_\$ \xleftarrow{unif} \mathcal{K}$
**return** $(c, K)$    **return** $(c, K_\$)$
```

Fig. 2: The multi-user $\mathsf{IND\text{-}CPA}$ experiment. Challenge oracle $\mathtt{Chall}$ can only be queried q_C many times and will return either the real key or a random key, depending on the challenge bit b.

[4] This will help streamline the security analysis of the protocol.

```
Exp $(n, q_C)-\mathsf{ANO\text{-}CPA}^{b=0}(\mathcal{A})$ | $(n, q_C)-\mathsf{ANO\text{-}CPA}^{b=1}$
for $j \in [n]$
    $(pk_{0,j}, sk_{0,j}) \leftarrow\$ \mathsf{KGen}$
    $(pk_{1,j}, sk_{1,j}) \leftarrow\$ \mathsf{KGen}$
$\boldsymbol{pk_0} \leftarrow (pk_{0,1}, \ldots, pk_{0,n})$
$\boldsymbol{pk_1} \leftarrow (pk_{1,1}, \ldots, pk_{1,n})$
$b' \leftarrow \mathcal{A}^{\mathtt{Chall}_{q_C}^{b=0}(.)}(\boldsymbol{pk_0}, \boldsymbol{pk_1})$
$b' \leftarrow \mathcal{A}^{\mathtt{Chall}_{q_C}^{b=1}(.)}(\boldsymbol{pk_0}, \boldsymbol{pk_1})$
return $b = b'$
```

```
Oracle $\mathtt{Chall}_{q_C}^{b=0}(j)$ | $\mathtt{Chall}_{q_C}^{b=1}(j)$
$(c, K) \leftarrow\$ \mathsf{Encap}(pk_{0,j})$
    $(c, K) \leftarrow\$ \mathsf{Encap}(pk_{1,j})$
return $(c, K)$
```

Fig. 3: The multi-user anonymity (ANO-CPA) experiment. The challenge oracle `Chall` can only be queried q_C many times and will either use the j-th public key in $\boldsymbol{pk_0}$ or the j-th key in $\boldsymbol{pk_1}$, depending on the challenge bit b.

```
Experiment $(n)-\mathsf{PKU}^{b=0}(\mathcal{A})$          | $(n)-\mathsf{PKU}^{b=1}$
for $j \in [n]$                                       | for $j \in [n]$
    $(pk_j, sk_j) \leftarrow\$ \mathsf{KGen}$           |     $pk_{j\$} \xleftarrow{unif} \mathcal{PK}$
$\boldsymbol{pk} \leftarrow (pk_1, \ldots, pk_n)$        | $\boldsymbol{pk}_\$ \leftarrow (pk_{1\$}, \ldots, pk_{n\$})$
$b' \leftarrow \mathcal{A}(\boldsymbol{pk})$            | $b' \leftarrow \mathcal{A}(\boldsymbol{pk}_\$)$
return $b = b'$
```

Fig. 4: The multi-user public-key uniformity (PKU) experiment.

Definition 6 (Multi-user security notions of KEM). *For an adversary $\mathcal{A}$, we define the advantages as:*

$$\mathbf{Adv}_{\mathsf{KEM}}^{(n,q_C)-\mathsf{IND\text{-}CPA}}(\mathcal{A}) := |\Pr[(n,q_C)-\mathsf{IND\text{-}CPA}^{b=0}(\mathcal{A})] - \Pr[(n,q_C)-\mathsf{IND\text{-}CPA}^{b=1}(\mathcal{A})]|, \textit{ and}$$

$$\mathbf{Adv}_{\mathsf{KEM}}^{(n,q_C)-\mathsf{ANO\text{-}CPA}}(\mathcal{A}) := |\Pr[(n,q_C)-\mathsf{ANO\text{-}CPA}^{b=0}(\mathcal{A})] - \Pr[(n,q_C)-\mathsf{ANO\text{-}CPA}^{b=1}(\mathcal{A})]|,$$

$$\mathbf{Adv}_{\mathsf{KEM}}^{(n)-\mathsf{PKU}}(\mathcal{A}) := |\Pr[(n)-\mathsf{PKU}^{b=0}(\mathcal{A})] - \Pr[(n)-\mathsf{IND\text{-}CPA}^{b=1}(\mathcal{A})]|$$

As was shown in [GKP18] (there Lemma 3.2), the advantage of an adversary $\mathcal{A}$ in a multi-user indistinguishability setting can be upper bounded by the advantage of an adversary $\mathcal{B}$ in the single-user setting, factored with the number of users n and the number of queries q_C. (Looking ahead, this loss will reflect the session-guessing loss in our PAKE proof when replacing multi-user notions with single-user notions.) Formally we have:

$$\mathbf{Adv}_{\mathsf{KEM}}^{(n,q_C)-\mathsf{IND\text{-}CPA}}(\mathcal{A}) \leq n \cdot q_C \cdot \mathbf{Adv}_{\mathsf{KEM}}^{\mathsf{IND\text{-}CPA}}(\mathcal{B}).$$

Below, we describe a generic relation between multi-user/challenge anonymity and single-user anonymity. The generic bound given here may be improved for KEMs whose underlying structure allow for a tighter reduction.

Theorem 1 (Multi-User anonymity from Single-User anonymity). *Let* KEM *be a key encapsulation mechanism. For any* $(n, q_C) - \mathsf{ANO\text{-}CPA}$ *adversary* $\mathcal{A}$ *against* KEM*, there exists an* $\mathsf{ANO\text{-}CPA}$ *adversary* $\mathcal{B}$ *against* KEM *such that*

$$\mathbf{Adv}_{\mathsf{KEM}}^{(n,\,q_C)\,-\,\mathsf{ANO\text{-}CPA}}(\mathcal{A}) \le n \cdot q_C \cdot \mathbf{Adv}_{\mathsf{KEM}}^{\mathsf{ANO\text{-}CPA}}(\mathcal{B}).$$

and the running time of $\mathcal{B}$ *is about that of* $\mathcal{A}$.

Proof (Proof of Theorem 1). We reduce $\mathsf{ANO\text{-}CPA}$ security of KEM to multi-user security anonymity $(n, q_C) - \mathsf{ANO\text{-}CPA}$, using a hybrid argument. Let $\mathcal{A}$ be an adversary in the $(n, q_C) - \mathsf{ANO\text{-}CPA}$ experiment in figure 3, with advantage $\mathbf{Adv}_{\mathsf{KEM}}^{(n,\,q_C)\,-\,\mathsf{ANO\text{-}CPA}}(\mathcal{A})$.

Consider the sequence of hybrid games $\mathbf{G}_{j,i}$ that successively changes the game for $\mathcal{A}$ from $b = 1$ (all challenges built using $\boldsymbol{pk_1}$) to $b = 0$ (all challenges built using $\boldsymbol{pk_0}$): In game $\mathbf{G}_{j,i}$, oracle $\mathtt{Chall}(j')$ uses

- the respective public key $pk_{0,j'}$ from $\boldsymbol{pk_0}$ if $j' < j$ or if $j' = j$ and $i' < i$, where i' is the number of the query to $\mathtt{Chall}(j')$,
- the respective public key $pk_{1,j'}$ from $\boldsymbol{pk_1}$ if $j' > j$ or if $j' = j$ and $i' \ge i$.

Using the triangle inequality yields

$$\begin{aligned}\mathbf{Adv}_{\mathsf{KEM}}^{(n,\,q_C)\,-\,\mathsf{ANO\text{-}CPA}}(\mathcal{A}) &= \left|\sum_{j=0}^{n-1}\sum_{i=0}^{q_C-1} \Pr[\mathbf{G}_{j,i}^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{j,i+1}^{\mathcal{A}} \Rightarrow 1]\right| \\ &\le \sum_{j=0}^{n-1}\sum_{i=0}^{q_C-1} \left|\Pr[\mathbf{G}_{j,i}^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{j,i+1}^{\mathcal{A}} \Rightarrow 1]\right| \ .\end{aligned}$$

where we made the convention that $\mathbf{G}_{j,q_C} := \mathbf{G}_{j+1,q_0}$ to handle index wrap-arounds.

We now give single-user $\mathsf{ANO\text{-}CPA}$ adversaries $\mathcal{B}_{ji}$ to upper bound the summands $\left|\Pr[\mathbf{G}_{j,i}^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{j,i+1}^{\mathcal{A}} \Rightarrow 1]\right|$: $\mathcal{B}_{ji}$ receives a single set of two challenge public keys, a ciphertext, and an encapsulated key, so (pk_0, pk_1, c^*, K^*), from its $\mathsf{ANO\text{-}CPA}$ challenger. $\mathcal{B}_{ji}$ will use its challenge input to simulate either $\mathbf{G}_{j,i}$ or $\mathbf{G}_{j,i+1}$: $\mathcal{B}_{ji}$ generates 2 vectors of $n-1$ many public keys $\boldsymbol{pk_0}, \boldsymbol{pk_1}$ using KGen and turns them into vectors of length n by inserting its own challenge public keys at the j-th position. $\mathcal{B}_{ji}$ then runs $\mathcal{A}$ on input $\boldsymbol{pk_0}, \boldsymbol{pk_1}$ and answers $\mathcal{A}$'s challenge queries as follows: upon the i'-th query to $\mathtt{Chall}(j')$, $\mathcal{B}_{ji}$ responds with

- a challenge constructed using the respective public key $pk_{0,j'}$ from $\boldsymbol{pk_0}$ if $j' < j$, or if $j' = j$ and $i' < i$
- a challenge constructed using the respective public key $pk_{1,j'}$ from $\boldsymbol{pk_1}$ if $j' > j$, or if $j' = j$ and $i' > i$
- its own challenge (c^*, K^*) if $j' = j$ and $i' = i$

When $\mathcal{A}$ outputs a guess to $\mathcal{B}$, $\mathcal{A}$ forwards the guess to its own challenger.

Since $\mathcal{B}_{ji}$ perfectly simulates $\mathbf{G}_{j,i}$ if its own challenge bit is 1, and $\mathbf{G}_{j,i+1}$ if its own challenge bit is 0, we have

$$\left|\Pr[\mathbf{G}_{j,i}^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{j,i+1}^{\mathcal{A}} \Rightarrow 1]\right| \le \mathbf{Adv}_{\mathsf{KEM}}^{\mathsf{ANO}}(\mathcal{B}_{ji}) \ .$$

Upper bounding $\left|\Pr[\mathbf{G}_{j,i}^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{j,i+1}^{\mathcal{A}} \Rightarrow 1]\right|$ accordingly yields

$$\sum_{j=0}^{n-1}\sum_{i=0}^{q_C-1} \left|\Pr[\mathbf{G}_{j,i}^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{j,i+1}^{\mathcal{A}} \Rightarrow 1]\right| \leq \sum_{j=0}^{n-1}\sum_{i=0}^{q_C-1} \left|\mathbf{Adv}_{\mathsf{KEM}}^{\mathsf{ANO}}(\mathcal{B}_{ji})\right| = n \cdot q_C \cdot \mathbf{Adv}_{\mathsf{KEM}}^{\mathsf{ANO}}(\mathcal{B}) \ ,$$

where $\mathcal{B}$ stems from folding the adversaries $\mathcal{B}_{ji}$ into a single one.

Theorem 2 (Multi-User Public-Key Uniformity from Single-User Public-Key Uniformity). *Let* KEM *be a key encapsulation mechanism. For any* $n-\mathsf{PKU}$ *adversary* $\mathcal{A}$ *against* KEM*, there exists an* PKU *adversary* $\mathcal{B}$ *against* KEM *such that*

$$\mathbf{Adv}_{\mathsf{KEM}}^{n,q_C\,-\,\mathsf{PKU}}(\mathcal{A}) \leq n \cdot \mathbf{Adv}_{\mathsf{KEM}}^{\mathsf{PKU}}(\mathcal{B}).$$

and the running time of $\mathcal{B}$ *is about that of* $\mathcal{A}$.

Proof (Proof of Theorem 2). We reduce PKU security of KEM to multi-user security anonymity $n-\mathsf{PKU}$, using a hybrid argument. Let $\mathcal{A}$ be an adversary in the $n-\mathsf{PKU}$ experiment in figure 4, with advantage $\mathbf{Adv}_{\mathsf{KEM}}^{n\,-\,\mathsf{PKU}}(\mathcal{A})$.

Consider the sequence of hybrid games $\mathbf{G}_i$ that successively changes the game for $\mathcal{A}$ from $b = 1$ (all public keys generated using KGen) to $b = 0$ (all public keys sampled uniformly): In game $\mathbf{G}_i$, the first i many public keys in $\boldsymbol{pk}$ are sampled using KGen, and the last $n-i$ many are sampled uniformly at random from $\mathcal{PK}$.

Using the triangle inequality yields

$$\begin{aligned}\mathbf{Adv}_{\mathsf{KEM}}^{n\,-\,\mathsf{PKU}}(\mathcal{A}) &= \left|\sum_{i=0}^{n-1} \Pr[\mathbf{G}_i^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{i+1}^{\mathcal{A}} \Rightarrow 1]\right| \\ &\leq \sum_{i=0}^{n-1} \left|\Pr[\mathbf{G}_i^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{i+1}^{\mathcal{A}} \Rightarrow 1]\right| \ .\end{aligned}$$

We now give single-user PKU adversaries $\mathcal{B}_i$ to upper bound the summands $\left|\Pr[\mathbf{G}_i^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{i+1}^{\mathcal{A}} \Rightarrow 1]\right|$: $\mathcal{B}_i$ receives a single public key pk^* from its PKU challenger. $\mathcal{B}_i$ will use its challenge input to simulate either $\mathbf{G}_i$ or $\mathbf{G}_{i+1}$: $\mathcal{B}_i$ generates a vector of $n-1$ many public keys $\boldsymbol{pk}$, using random sampling from $\mathcal{PK}$ for the first $i-1$ many, and KGen for the positions $i+1$ to $n-1$. It turns the vector into a vector of length n by inserting its own challenge public key at the i-th position.

When $\mathcal{A}$ outputs its guess to $\mathcal{B}_i$, $\mathcal{B}_i$ forwards the guess to its own challenger.

Since $\mathcal{B}_i$ perfectly simulates $\mathbf{G}_i$ if its own challenge bit is 1, and $\mathbf{G}_{i+1}$ if its own challenge bit is 0, we have

$$\left|\Pr[\mathbf{G}_i^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{i+1}^{\mathcal{A}} \Rightarrow 1]\right| \leq \mathbf{Adv}_{\mathsf{KEM}}^{\mathsf{ANO}}(\mathcal{B}_i) \ .$$

Upper bounding $\left|\Pr[\mathbf{G}_i^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{i+1}^{\mathcal{A}} \Rightarrow 1]\right|$ accordingly yields

$$\sum_{i=0}^{n-1} \left|\Pr[\mathbf{G}_i^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{i+1}^{\mathcal{A}} \Rightarrow 1]\right| \leq \sum_{i=0}^{n-1} \left|\mathbf{Adv}_{\mathsf{KEM}}^{\mathsf{ANO}}(\mathcal{B}_i)\right| = n \cdot \mathbf{Adv}_{\mathsf{KEM}}^{\mathsf{ANO}}(\mathcal{B}) \ ,$$

where $\mathcal{B}$ stems from folding the adversaries $\mathcal{B}_i$ into a single one.

2.3 Notation

We denote the probability of adversary $\mathcal{A}$ winning in game $\mathbf{G}_i$ as $\mathbf{Adv}_i$.

3 The Protocol PAKEM

Initiator (Client)	**PAKEM Protocol**	**Responder (Server)**
Password pw		Password pw
	Transmit Encrypted Public Key	
$k_{pw} \leftarrow \texttt{KDF}(pw)$		$k_{pw} \leftarrow \texttt{KDF}(pw)$
$(pk, sk) \xleftarrow{\$} \mathsf{KGen}$		
$apk \leftarrow \texttt{BC.enc}(k_{pw}, pk)$	$\xrightarrow{apk}$	$pk' \leftarrow \texttt{BC.dec}(k_{pw}, apk)$
	Establish Session Pre-Key	
$K' \leftarrow \mathsf{Decap}(sk, c)$	$\xleftarrow{c}$	$(c, K) \xleftarrow{\$} \mathsf{Encap}(pk')$
	$\xleftarrow{tag_r}$	$tag_r \leftarrow \texttt{H}(pw, apk, pk', c, K, "r")$
$tag_i \leftarrow \texttt{H}(pw, apk, pk, c, K', "i")$	$\xrightarrow{tag_i}$	
	Key Confirmation, Key Derivation	
$tag'_r \leftarrow \texttt{H}(pw, apk, pk, c, K', "r")$		$tag'_i \leftarrow \texttt{H}(pw, apk, pk', c, K, "i")$
`if` $tag'_r = tag_r$		`if` $tag'_i = tag_i$
$SK \leftarrow \texttt{KDF}'(tag_r, K')$		$SK \leftarrow \texttt{KDF}'(tag_r, K)$
`output` SK `and accept`		`output` SK `and accept`
`terminate`		`terminate`

Fig. 5: The PAKEM protocol, using a key encapsulation mechanism $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap})$ and a block cipher $\texttt{BC} = (\texttt{BC.enc}, \texttt{BC.dec})$ that is modeled as an ideal cipher in the proof. Messages c and tag_i are part of the same round, so can be transmitted together, making this a three-round protocol.

We describe a 3-message password-authenticated key exchange protocol based on an ideal cipher IC and an implicitly-rejecting key encapsulation mechanism KEM in figure 5. The protocol achieves mutual authentication and AKE security according to the BPR model, which we prove in section 5. Two parties, the initiator and the responder, share a common password pw and proceeds in three phases: first, the initiator will generate a KEM key pair, encrypt the public key using the password, and send it to the responder. The responder uses the password to recover the public key, then computes an encapsulation and pre-key. The responder sends that encapsulation to the initiator, along with a responder tag[5]. The initiator decapsulates the ciphertext to obtain a pre-key and compares the tag to one derived from its own state. The initiator only

[5] Because KEM is implicitly rejecting and therefore cannot not strongly robust, this tag is needed to authenticate the responder.

outputs a session key derived from the pre-key if the tags match. Finally, the initiator computes an initiator tag and sends it to the responder. The responder checks this tag against its own state and outputs a session key derived from its pre-key if it matches. Under the correctness of KEM, the session key output by both parties is identical if and only if both parties used the same password.

4 Security Model

Our security analysis is based on the BPR model for authenticated key exchange [BPR00]. In the BPR model, an adversary can interact with the oracles shown in Fig. 6.

Query	Return Value	Description
`Execute`(P, i, P', j)	(apk, c, tag_r, tag_i)	Passive attack: Return transcript of an honest protocol execution between parties P and P', using the ith/jth session of P/P'
`Send`$(P, i, msg, flow)$	msg'	Active attack: Send message msg to the oracle representing honest user P, causing it to proceed depending on its state. Flow indicator enumerates the messages in a run of the protocol and improves readability of the oracle.
`Reveal`(P, i)	SK$[P, i]/\perp$	Session key leakage: Return session key SK of (P, i) iff (P, i) terminated, else $\perp$; marks this instance and its matching instance "unfresh"
`Corrupt`(P, pw)	PWD$[P, :]$	Password leakage or overwrite: Either return dictionary of passwords PWD$[P, :]$ held by party P, or allow adversary to overwrite password dictionary with PWD'$[P, :]$.
`Test`(P, i)	SK$[P, i]/SK^{\$}$	Session key challenge: Attack ith session of party P. Only for terminated instances.
`KDF`(pw)	k_{pw}	Random oracle, input password $pw \in$ PWD, output Ideal Cipher key k_{pw}
`H`(msg)	tag	Random oracle, input message msg, output $tag \in \mathcal{T}$
`KDF'`(msg)	SK	Random oracle, input message msg, output session key $SK \in \mathcal{SK}$
`IC.enc`(k, m)	c	Ideal cipher encryption on input (key, message)
`IC.dec`(k, c)	m	Ideal cipher decryption on input (key, ciphertext)

Fig. 6: Overview of the PAKE adversary's oracles provided by the security game. Top part (above double midrule): oracles present in the BPR model. Bottom part: Random oracles and ideal cipher oracles to which the attacker additionally has access when attacking the PAKEM protocol, as our security proof idealizes the hash functions and the block cipher used in PAKEM. The model also includes an *Initialization* phase, where passwords are generated by some long-lived-key generator PW.

Definition 7 (Partnering). *Two instances P_i, P'_j are partnered iff both instances have terminated (i.e. reached a `terminate` instruction) with the same transcript and session key.*

Intuitively, 'unfreshness' expresses that the adversary may have learned the to-be-tested session's key SK in a trivial way, i.e., by having interacted with the oracles revealing secret information in a way such that SK becomes derivable regardless of the nature of the protocol. Concretely, the cases we cover in our freshness definition below are a), simply requesting the key from the `Reveal` oracle, and b), learning a password pw via `Corrupt` and then actively interfering with the test session, e.g., using pw to manipulate the peer into using a session key of the adversary's choosing.

Definition 8 (Freshness with forward secrecy). *Suppose that the adversary made exactly one `Test` query, and it was to party P and instance i. We say the oracle of instance i of party P is **unfresh** if at any time, there was a `Reveal` query to instance (P, i) or the instance (P', j) that it is partnered with.*

*We also say it is **unfresh** if both the following two conditions hold:*
- *Before the `Test` query, there was a `Corrupt` query on the test session's holder P or its peer.*
- *One of the messages sent to P concerning the test session was* manipulated *by the adversary, i.e., there was a `Send(`P, i`)` query.*

*The oracle of (P, i) is only considered **fresh** if neither of these conditions are met.*

Definition 9 (Key-Indistinguishability of Authenticated Key Exchange). *For a protocol Π and an adversary $\mathcal{A}$, we define $\mathcal{A}$'s advantage with respect to Π as $\mathbf{Adv}_{\Pi}^{\mathsf{BPR}}(\mathcal{A})$. We say that the adversary with access to the oracles in figure 6* wins *if on issuing a `Test` query for a party (P, i) that has terminated and is fresh (see 8), $\mathcal{A}$ correctly guesses the bit selected in the test query. The security experiment is shown in figure 7 The advantage of $\mathcal{A}$ is then defined as*

$$\mathbf{Adv}_{\Pi}^{\mathsf{BPR}}(\mathcal{A}) := |\Pr[Exp_{\Pi}^{\mathsf{BPR}}(\mathcal{A})]| - 1/2$$

```
Experiment $\mathrm{Exp}_{\Pi}^{\mathsf{BPR}}(\mathcal{A})$
$b \xleftarrow{unif} \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^b}(\mathcal{P}, N, \mathcal{D})$
return $b = b'$
```

Fig. 7: The BPR security game for active adversaries. $\mathcal{O}^b$ = indicates the collection of oracles {`Execute`, `KDF`, `H`, `KDF'`, `IC.enc`, `IC.dec`, `Send`, `Reveal`, `Corrupt`,`Test`^{b}}. Here, $\mathcal{P}$ is the party set, N is the number of instances per party and $\mathcal{D}$ is the password dictionary.

Definition 10 (Explicit Mutual Authentication). *A protocol achieves explicit mutual authentication if parties* accept *if and only if there exists a partnered party that accepts with the same output.*

Protocol PAKEM uses tags in both directions, and while the responder tag is needed for BPR security of the protocol, the initiator tag's purpose is to achieve explicit mutual authentication. This approach follows the AddCSA (add client-to-server authentication) transformation introduced in [BPR00].

5 Security of PAKEM

Essentially, our main Theorem 3 below relates forward security of PAKEM to security of the used KEM, in the Random Oracle Model and the Ideal Cipher model. During its proof, we will consider an adversary playing the BPR security game for our protocol PAKEM. The adversary has access to

– oracles `Execute`, `Send`, `Reveal`, `Corrupt`, and `Test`, all of which we make explicit in figure 9, as well as
– random oracles `KDF`, `KDF'`, and `H` and
– ideal cipher oracles `IC.enc` and `IC.dec`.

Theorem 3 (Tight security of PAKEM in the Random Oracle and Ideal Cipher models from Multi-User Security of KEM). *Let KEM be a key encapsulation mechanism that is $(1-\delta)$-correct, let KDF, KDF', and H be modeled as random oracles, BC be modeled as an ideal cipher, and let $\mathcal{A}$ be a BPR adversary against PAKEM[KEM, KDF, KDF', H, BC], issuing at most q_S many Send queries (i.e. active attacks), q_E many Execute (number of transcripts the adversary can see), $q_{IC.dec}$ many decryption queries to the ideal cipher, q_{IC} many queries to the ideal cipher in total (encryption or decryption), and q_H many queries to its respective random oracles. Then there exist multi-user-IND-CPA adversary $\mathcal{B}^{\mathsf{I}}$ a multi-user-ANO-CPA adversary $\mathcal{B}^{\mathsf{A}}$ and a multi-user-PKU adversary $\mathcal{B}^{\mathsf{U}}$ against KEM such that*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{BPR}}_{\mathsf{PAKEM}}(\mathcal{A}) \leq \frac{q_S}{|\mathcal{D}|} &+ \mathbf{Adv}^{(q_{IC.dec}+q_S+q_E)-\mathsf{PKU}}_{\mathsf{KEM}}(\mathcal{B}^{\mathsf{U}}) + 2\mathbf{Adv}^{(q_S+q_E,\,q_S+1)-\mathsf{ANO\text{-}CPA}}_{\mathsf{KEM}}(\mathcal{B}^{\mathsf{A}}) \\ &+ 2\mathbf{Adv}^{(q_S+q_E,\,q_S+1)-\mathsf{IND\text{-}CPA}}_{\mathsf{KEM}}(\mathcal{B}^{\mathsf{I}}) + \frac{3\cdot q_{IC}^2\cdot}{2\cdot|\mathcal{PK}|} + (q_S+q_E)\cdot\delta \\ &+ q_{RO}\cdot(q_S+q_E)\cdot(\frac{1}{|\mathcal{SK}|}+\frac{1}{|\mathcal{K}|}) + q_{RO}^2\cdot(\frac{1}{2\cdot|T|}+\frac{1}{2\cdot|\mathcal{K}_{pw}|})\end{aligned}$$

and the running time of $\mathcal{B}^{\mathsf{I}}$, $\mathcal{B}^{\mathsf{A}}$, and $\mathcal{B}^{\mathsf{U}}$ is about that of $\mathcal{A}$.

Theorem 4 (Security of PAKEM in the Random Oracle and Ideal Cipher Models from Single-User Security of KEM). *Let KEM be a key encapsulation mechanism that is $(1-\delta)$-correct, let KDF, KDF', and H be modeled as random oracles, BC be modeled as an ideal cipher, and let $\mathcal{A}$ be a BPR adversary against PAKEM[KEM, KDF, KDF', H, BC], issuing at most q_S many Send queries (i.e. active attacks), q_E many Execute (number of transcripts the adversary can see), $q_{IC.dec}$ many decryption queries to the ideal cipher, q_{IC} many queries to the ideal cipher in total (encryption or decryption), and $q_{KDF}/q_{KDF'}/q_H$ many queries to its respective random oracles. Then there exist multi-user-IND-CPA adversary $\mathcal{B}^{\mathsf{I}}$ a multi-user-ANO-CPA adversary $\mathcal{B}^{\mathsf{A}}$ and a multi-user-PKU adversary $\mathcal{B}^{\mathsf{U}}$ against KEM such that*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{BPR}}_{\mathsf{PAKEM}}(\mathcal{A}) \leq \frac{q_S}{|\mathcal{D}|} &+ (q_{IC.dec}+q_S+q_E)\cdot\mathbf{Adv}^{\mathsf{PKU}}_{\mathsf{KEM}}(\mathcal{B}^{\mathsf{U}}) + 2\cdot(q_S+q_E)\cdot(q_S+1)\cdot\mathbf{Adv}^{\mathsf{IND}}_{\mathsf{KEM}}(\mathcal{B}^{\mathsf{I}}) \\ &+ 2\cdot(q_{IC.dec}+q_S+q_E)\cdot(q_S+1)\cdot\mathbf{Adv}^{\mathsf{ANO}}_{\mathsf{KEM}}(\mathcal{B}^{\mathsf{A}}) + (q_S+q_E)\cdot\delta \\ &+ q_{RO}\cdot(q_S+q_E)\cdot(\frac{1}{|\mathcal{SK}|}+\frac{1}{|\mathcal{K}|}) + q_{RO}^2\cdot(\frac{1}{2\cdot|T|}+\frac{1}{2\cdot|\mathcal{K}_{pw}|}) + \frac{3\cdot q_{IC}^2\cdot}{2\cdot|\mathcal{PK}|}\end{aligned}$$

and the running time of $\mathcal{B}^{\mathsf{I}}$, $\mathcal{B}^{\mathsf{A}}$, and $\mathcal{B}^{\mathsf{U}}$ is about that of $\mathcal{A}$.

Theorem 4 follows directly from plugging the generic multi-user reductions in 2.2 into Theorem 3, we therefore proceed by proving the 'tight' theorem Theorem 3.

Intuitively, the security proof has three security goals: Showing that

(SG1) the adversary can test at most one password per session with which it actively interferes,
(SG2) honest protocol runs do not leak significant amounts of information on the password, and
(SG3) the session key looks independent of both session transcript and password to the adversary unless it manages to attack the underlying KEM.

Pseudo-code for the oracles is shown in figure 9. Amongst the other oracles, Fig. 9 also sketches the `Send` oracle. In `Send`, we use a flow identifier to separate the different stages of the protocol- We make the convention that oracle `Send` will only proceed if it is in the correct state for the received message: for example, if an instance receives a ciphertext c without having received a flow-0 message that caused it to generate a key pair, it will not respond. As shown in figure 9, we at first will also model the `Execute` oracle using the `Send` oracle.The adversary can query the `Test` oracle exactly once, for a party and instance that fulfills the freshness definition.

High-level overview of proof. In the security proof, we will argue that for every actively manipulated session, we can uniquely determine which password was tested by the adversary. This implies that the adversary can use at most one password per active attack, satisfying security goal SG1. During that argument, we need to catch the side-case that protocol messages could stem from multiple passwords due to collisions. Game hops $\mathbf{G}_1$ to $\mathbf{G}_4$ are aimed at eliminating these collisions.
After that, we address security goals SG2 and SG3 by eliminate leakage on the password and the session key with game hops $\mathbf{G}_7$ to $\mathbf{G}_9$. Game hops $\mathbf{G}_6$ and $\mathbf{G}_5$ are in preparation for these changes.

Game	Change	Loss
$\mathbf{G}_1$	KDF collisions	$\frac{q_{\mathtt{KDF}}^2}{2 \cdot \lvert\mathcal{K}_{pw}\rvert}$
$\mathbf{G}_2$	IC simulation by sampling	$\frac{q_{\mathtt{IC}}^2}{2 \cdot \lvert\mathcal{PK}\rvert}$
$\mathbf{G}_3$	IC collisions	$\frac{q_{\mathtt{IC}}^2}{\lvert\mathcal{PK}\rvert}$
$\mathbf{G}_4$	abort on responder tag collision	$\frac{q_{\mathtt{H}}^2}{2 \cdot \lvert T\rvert}$
$\mathbf{G}_5$	sample IC outputs using KGen	$\mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathtt{IC.dec}} + q_{\mathtt{S}} + q_{\mathtt{E}}) - \mathsf{PKU}}$
$\mathbf{G}_6$	detect corr pw	0
$\mathbf{G}_7$	randomize pk	$\mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathtt{S}} + q_{\mathtt{E}}, q_{\mathtt{S}} + 1) - \mathsf{ANO\text{-}CPA}}$
$\mathbf{G}_8$	randomize pre-key	$\mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathtt{S}} + q_{\mathtt{E}}, q_{\mathtt{S}} + 1) - \mathsf{IND\text{-}CPA}} + (q_{\mathtt{S}} + q_{\mathtt{E}}) \cdot \delta$
$\mathbf{G}_9$	randomize responder tag	$\frac{q_{\mathtt{H}} \cdot (q_{\mathtt{S}} + q_{\mathtt{E}})}{\lvert\mathcal{K}\rvert}$
$\mathbf{G}_{10}$	randomize session key	$\frac{q_{\mathtt{KDF'}} \cdot (q_{\mathtt{S}} + q_{\mathtt{E}})}{\lvert\mathcal{SK}\rvert}$
$\mathbf{G}_{10}$	abort on corr pw	$\frac{q_{\mathtt{S}}}{\lvert\mathcal{D}\rvert} + \Delta \Pr[\mathtt{corrPW}]$

Fig. 8: Overview of all game changes and their associated loss.

Oracle $\texttt{Send}(P, i, msg, flow)$

```
if CORRUPT[{P,PART(P,i)}]
  FRESH[$(P,i)$] $\leftarrow$ false
  FRESH[$\{P, PART(P,i)\}$] $\leftarrow$ false
if $flow = 0$        //prompt first message
  $k_{pw} \leftarrow \texttt{KDF}(pw)$
  $(pk, sk) \leftarrow\$ \mathsf{KGen}$
  $apk \leftarrow \texttt{IC.enc}_{k_{pw}}(pk)$
  return apk
if $flow = 1$        //first protocol message
  $apk \xleftarrow{\texttt{parse}} msg$
  $k_{pw} \leftarrow \texttt{KDF}(pw)$
  $pk' \leftarrow \texttt{IC}^{-1}_{k_{pw}}(apk)$
  $(c, K) \leftarrow\$ \mathsf{Encap}(pk')$
  $tag_i \leftarrow \texttt{H}(pw, apk, pk', c, K, "r")$
  return $(c, tag_i)$
if $flow = 2$        //second protocol message
  $c, tag_i \xleftarrow{\texttt{parse}} msg$
  $K' \leftarrow \mathsf{Decap}(sk, c)$
  if $tag_i = \texttt{H}(pw, apk, pk, c, K', "r")$
    $SK \leftarrow \texttt{KDF}'(tag_i, K')$
    K[(P,i)]$\xleftarrow{\texttt{set}} SK$
    $tag_r \leftarrow \texttt{H}(pw, apk, pk, c, K', "i")$
    return $tag_r$
  else return $\perp$
if $flow = 3$        //third protocol message
  $tag_r \xleftarrow{\texttt{parse}} msg$
  if $tag_r = \texttt{H}(pw, apk, pk', c, K, "i")$
    $SK \leftarrow \texttt{KDF}'(tag_i, K)$
    K[(P,i)]$\xleftarrow{\texttt{set}} SK$
```

Initialization

```
for $(P, P') \in \mathcal{P} \times \mathcal{P}$
  $pw \leftarrow\$ \mathrm{PWD}()$
  PWD[$\{P, P'\}$] $\xleftarrow{\texttt{set}} pw$
```

Oracle $\texttt{Execute}(P, i, P', j)$

```
$apk \leftarrow \texttt{Send}(P, i, \perp, 0)$
$c, tag_i \leftarrow \texttt{Send}(P', j, apk, 1)$
$tag_r \leftarrow \texttt{Send}(P, i, (c, tag_i), 2)$
$\texttt{Send}(P', j, tag_r, 3)$
return $(apk, c, tag_i, tag_r)$
```

Oracle $\texttt{Corrupt}(P, \text{PWD'})$

```
$\text{PWD}_P \leftarrow \text{PWD}[\{P, :\}]$
CORRUPT[{P,:}]$\leftarrow$ true            //all partners
for $P' \in \mathcal{P}$                   //replace passwords
  if PWD'[$\{P, P'\}$]$\neq\perp$
    PWD[$\{P, P'\}$] $\xleftarrow{\texttt{set}}$PWD'[$\{P, P'\}$]
return $\text{PWD}_P$
```

Oracle $\texttt{Reveal}(P, i)$

```
FRESH[$(P,i)$] $\leftarrow$ false          //this instance
return SK[$(P,i)$]
```

Oracle $\texttt{Test}^{b=0}(P, i)$ $\texttt{Test}^{b=1}(P, i)$

```
if not (FRESH[$(P,i)$] and FRESH[PART$(P,i)$])
  return $\perp$
else if SK[$(P,i)$] $=\perp$               //check if terminated
  return $\perp$
else:
                                           $SK_\$ \xleftarrow{unif} \mathcal{SK}$
  return $SK$                              return $SK_\$$
```

Fig. 9: The oracles in the security game for PAKEM. PWD is the dictionary of passwords, CORRUPT and FRESH indicate the corruption status of a session and PART indicates the intended partner of an instance. Password generation in the initialization phase (**Initialization**) is modeled using the long-lived key generator PW.

Proof (Theorem 3).

5.1 Original Security Game

Game $\mathbf{G}_0$*: Original Game* Game $\mathbf{G}_0$ represents the original BPR security game. The `Send` and `Execute` oracles answer queries according to the protocol, as shown in figure 9.

5.2 Eliminating Collisions (SG1)

In a first step, we address collision events that would allow distinct passwords to result in the same transcript.

Game $\mathbf{G}_1$*: Abort on Key Derivation Function Collisions* First we address collisions in the key derivation function that would allow an adversary to use an ideal cipher key that corresponds to multiple passwords. Intuitively, this could mean that an adversary could use this derived key and succeed in an attack on a session even if a password that is not the correct one for this session is used. This attack applies to both to *malicious initiator* as well as *malicious responder* adversaries.[6]

We now keep a list of all previous queries to the `KDF` oracle by recording all input-output pairs (pw, k_{pw}). Let `KDFColl` be the event there were two queries to the `KDF` oracle s.t. for two distinct passwords $pw \neq pw'$, the derived keys are the same:

$$\texttt{KDFColl} : k_{pw} = k_{pw'} \text{ for queries } k_{pw} \leftarrow \texttt{KDF}(pw), k_{pw'} \leftarrow \texttt{KDF}(pw').$$

In game $\mathbf{G}_1$, we abort whenever this event occurs. Let $q_{\texttt{KDF}}$ be the number of queries to `KDF`. Since KDF is modeled as a random oracle, we can bound the probability of this event using a standard collision bound over the number of queries and the size of the output space of `KDF`: $\Pr[\texttt{KDFColl}] \leq \frac{q_{\texttt{KDF}}^2}{2 \cdot |\mathcal{K}_{pw}|}$. Since games $\mathbf{G}_0$ and $\mathbf{G}_1$ are identical unless `KDFColl` occurs, the distance of the adversary's success probability is bounded:

$$|\mathbf{Adv}_0 - \mathbf{Adv}_1| = \Pr[\texttt{KDFColl}] \leq \frac{q_{\texttt{KDF}}^2}{2 \cdot |\mathcal{K}_{pw}|}$$

From now on, we can argue that any password-derived key k_{pw} used in some protocol execution or oracle query corresponds to at most one password.

Game $\mathbf{G}_2$*: Simulate Ideal Cipher* We now simulate a "modified" ideal cipher by lazy sampling where instead of choosing an output from the set of remaining outputs, we sample one from the entire domain and abort in case we sample a value that would violate the permutation property. This is done in preparation for games $\mathbf{G}_5$ and $\mathbf{G}_7$, where we replace ideal cipher outputs with public keys generated using KGen. This is illustrated in figure Fig. 10.

Game $\mathbf{G}_2$ is identical to $\mathbf{G}_1$ unless it aborts in line 5 of figure Fig. 10. The probability of this occurring can be bounded using a standard collision bound in the total number of ideal cipher queries $q_{\texttt{IC}}$ and the size of the public key space $|\mathcal{PK}|$ and therefore:

$$|\mathbf{Adv}_1 - \mathbf{Adv}_2| \leq \frac{q_{\texttt{IC}}^2}{2 \cdot |\mathcal{PK}|}.$$

[6] We will later show the relation between more complex adversaries, such as those executing machine-in-the-middle attacks and an adversary and these two base types.

```
IC.enc($k_{pw}, pk$)
1 if $\exists$ record $(k_{pw}, pk, apk)$
2   return $apk$
3 else
4   $apk' \xleftarrow{unif} \mathcal{PK}$
5   if $\exists$ record $(k_{pw}, \star, apk')$: abort
6   create record $(k_{pw}, pk, apk')$
7   return $apk'$
```

```
IC.dec($k_{pw}, apk$)
1 if $\exists$ record $(k_{pw}, pk, apk)$
2   return $pk$
3 else
4   $pk' \xleftarrow{unif} \mathcal{PK}$
5   if $\exists$ record $(k_{pw}, pk', \star)$: abort
6   create record $(k_{pw}, pk', apk)$
7   return $pk'$
```

Fig. 10: The simulated ideal cipher with abort.

Game $\mathbf{G}_3$*: Abort on Ideal Cipher Collisions* Next we eliminate collisions in the ideal cipher. Collisions can allow the adversary to test multiple passwords in a single session, violating security goal (SG1). The probability of such collisions occurring is therefore directly relevant to the security of the scheme. There are two types of collision for which this is the case.

The *first* type of collision occurs if the adversary finds that some public key and authenticated public key are mapped to each other under two distinct passwords. Formally, this would imply that there were two queries to the ideal cipher such that for $k_{pw} \neq k'_{pw}$

$$\begin{aligned}\texttt{ICColl1} : (pk, apk) = (pk', apk') \text{ for two queries:}\\ \big(\textbf{either } apk \leftarrow \texttt{IC.enc}(k_{pw}, pk) \ \textbf{ or } pk \leftarrow \texttt{IC.dec}(k_{pw}, apk)\big)\\ \textbf{and } \big(\textbf{either}(apk' \leftarrow \texttt{IC.enc}(k'_{pw}, pk') \ \textbf{ or } pk' \leftarrow \texttt{IC.dec}(k'_{pw}, apk')\big)\end{aligned}$$

Therefore, an adversary sending this *apk* value to the Send oracle could test the passwords corresponding to k_{pw} and k'_{pw} in one query.

The *second* type of collision occurs if there were at least two ideal cipher *encryption* queries for *distinct* passwords and public keys that returned the same authenticated public key. Knowledge of *apk* with this property allows the adversary to test both passwords in one query.[7] Formally, this would imply that there were two queries to the ideal cipher such that for $k_{pw} \neq k'_{pw}$:

$$\texttt{ICColl2} : \ apk = apk' \text{ for queries: } apk \leftarrow \texttt{IC.enc}(k_{pw}, pk), apk' \leftarrow \texttt{IC.enc}(k'_{pw}, pk')$$

In game $\mathbf{G}_3$, we abort whenever ICColl1 or ICColl2 occur. We define the event ICColl where $\Pr[\texttt{ICColl}] := \Pr[\texttt{ICColl1} \vee \texttt{ICColl2}]$. We argue that the probability of this event is upper-bounded by a standard collision bound in the total number of ideal cipher queries (encryption and decryption) $q_{\texttt{IC}}$ and the size of the ideal cipher domain $|\mathcal{PK}|$, since it requires sampling. In game $\mathbf{G}_3$, we abort whenever ICColl occurs. Since games $\mathbf{G}_2$ and $\mathbf{G}_3$ are identical unless ICColl occurs, it holds that

$$|\mathbf{Adv}_2 - \mathbf{Adv}_3| = \Pr[\texttt{ICColl}] \leq \frac{q^2_{\texttt{IC}}}{2 \cdot |\mathcal{PK}|} + \frac{q^2_{\texttt{IC.enc}}}{2 \cdot |\mathcal{PK}|} \leq \frac{q^2_{\texttt{IC}}}{|\mathcal{PK}|}.$$

[7] To further elaborate, this would imply that for this *apk*, there are two keys $k_{pw} \neq k_{pw'}$ and therefore two passwords $pw \neq pw'$ for which the adversary could know the secret keys associated with the public keys $pk \neq pk'$. This would then allow the adversary to decrypt ciphertexts for both these public keys, to derive two candidate session keys. Either one of them could then be compared to the session key output by the Test query.

```
$\texttt{IC.enc}_{\mathbf{G}_2}(k_{pw}, pk)$    $\boxed{\texttt{IC.enc}_{\mathbf{G}_3}(k_{pw}, pk)}$
if $\exists$ record $(k_{pw}, pk, apk)$
  return $apk$
else
  $apk' \xleftarrow{unif} \mathcal{PK}$
  if $\exists$ record $(k_{pw}, \star, apk')$: abort
  if $\exists$ record $(\star, pk, apk')$: abort                 ICColl1
  if $\exists$ record $(\star, \star, apk', \texttt{"enc"})$: abort   ICColl2
  create record $(k_{pw}, pk, apk')$
  create record $(k_{pw}, pk, apk', \texttt{"enc"})$
  return $apk'$
```

```
$\texttt{IC.dec}_{\mathbf{G}_2}(k_{pw}, apk)$    $\boxed{\texttt{IC.dec}_{\mathbf{G}_3}(k_{pw}, apk)}$
if $\exists$ record $(k_{pw}, pk, apk)$
  return $pk$
else
  $pk' \xleftarrow{unif} \mathcal{PK}$
  if $\exists$ record $(k_{pw}, pk', \star)$: abort
  if $\exists$ record $(\star, pk', apk)$: abort                 ICColl1
  create record $(k_{pw}, pk', apk)$
  return $pk'$
```

Fig. 11: The simulated ideal cipher. In game $\mathbf{G}_3$, the game aborts whenever there is a collision in the ideal cipher that would allow the adversary to test two passwords.

Game $\mathbf{G}_4$*: Abort on Server Tag Collision* We now create a record for all queries to H by the adversary and let ROColl be the event that the random oracle outputs the same value twice, for different inputs, in which case game $\mathbf{G}_4$ aborts. The probability of ROColl occurring is bounded by a standard collision bound in the number of queries $q_{\texttt{H}}$ to H and the size of the tag space T: $\Pr[\texttt{ROColl}] \leq \frac{q_{\texttt{H}}^2}{2\cdot|T|}$. Since games $\mathbf{G}_3$ and $\mathbf{G}_4$ are identical unless ROColl or occurs, it holds that:

$$|\mathbf{Adv}_3 - \mathbf{Adv}_4| = \Pr[\texttt{ROColl}] \leq \frac{q_{\texttt{H}}^2}{2 \cdot |T|}.$$

For every responder tag output by the H, there is now exactly one password that was used to create it. Therefore, whenever a *malicious initiator* adversary submits such a tag, this tag corresponds to at most one password.

At this point, we have proven that it is unlikely for an adversary to be able to test multiple passwords in a single query, in accordance with security goal 1 (SG1).

Game $\mathbf{G}_5$*: Sample Ideal Cipher Outputs Using* KEM *Key Generation* In game $\mathbf{G}_5$, we replace the way the ideal cipher samples outputs. On decryption queries, instead of sampling from the output domain uniformly at random, we use the key generation algorithm of KEM. This is an auxiliary step that we do in preparation for the separation of ciphertexts c and the password, which we will do using the anonymity property of KEM in game $\mathbf{G}_7$. The change is depicted in figure Fig. 12.

We will now argue that an adversary noticing this change can be used to attack the n−key-uniformity ($n-\mathsf{PKU}$) property of the underlying KEM, by means of a reduction $\mathcal{B}^{\mathsf{U}}$. Let $\mathcal{A}$ be the adversary running either in game $\mathbf{G}_4$ or $\mathbf{G}_5$, issuing at most $q_{\texttt{IC.dec}}$ many queries to the ideal cipher decryption oracle. We define adversary $\mathcal{B}^{\mathsf{U}}$ against the $n-\mathsf{PKU}$ experiment (defined in Fig. 4) as follows (for the sake of formality, we give the pseudo-code of $\mathcal{B}^{\mathsf{U}}$ in Fig. 13):

```
$\texttt{IC.dec}_{\mathbf{G}_4}(k_{pw}, apk)$                    $\boxed{\texttt{IC.dec}_{\mathbf{G}_5}(k_{pw}, apk)}$
if $\exists$ record $(k_{pw}, pk, apk)$
  return $pk$
else
  $pk' \xleftarrow{unif} \mathcal{PK}$                          $\boxed{pk' \leftarrow\!\!\$\ \mathsf{KGen}}$
  if $\exists$ record $(k_{pw}, pk', \star)$: abort
  if $\exists$ record $(\star, pk', apk)$: abort
  create record $(k_{pw}, pk', apk)$
  return $pk'$
```

Fig. 12: The simulated ideal cipher now samples using the KEM's key generation algorithm KGen instead of uniformly at random from the domain $\mathcal{PK}$.

$\mathcal{B}^{\mathsf{U}}$ receives a vectors of challenge public keys $\boldsymbol{pk}$ of dimension n from its $n-\mathsf{PKU}$ challenger, where $n := q_{\texttt{IC.dec}} + q_{\mathsf{S}} + q_{\mathsf{E}}$. (Depending on the challenger's bit b_{PKU}, $\boldsymbol{pk}$ is generated using KGen or drawn uniformly at random from $\mathcal{PK}$.)

$\mathcal{B}^{\mathsf{U}}$ samples an own challenge bit b', runs $\mathcal{A}$ and answers $\mathcal{A}$'s queries to the Oracles $\texttt{H}$, $\texttt{IC.enc}$, $\texttt{KDF}$, $\texttt{Reveal}$, and $\texttt{Test}^{b'}$ according to the oracles in $\mathbf{G}_4$.

When simulating ideal cipher decryption queries, $\mathcal{B}^{\mathsf{U}}$ embeds the challenge public keys stemming from its input vector $\boldsymbol{pk}$: Upon a query to oracle $\texttt{IC.dec}$, instead of sampling an output like game $\mathbf{G}_4$ (see line 4 of Fig. 10), $\mathcal{B}^{\mathsf{U}}$ uses the next value in $\boldsymbol{pk}$. (Note that the changes in game 2 ensures that the resulting output is still a permutation.) In case a query is repeated, it repeats the respective public key.

When $\mathcal{A}$ outputs a guess b, $\mathcal{B}^{\mathsf{U}}$ checks if $b = b'$ and in that case returns $b'_{\mathsf{PKU}} := 1$ as its own output bit, otherwise, $\mathcal{B}^{\mathsf{U}}$ returns $b'_{\mathsf{PKU}} := 0$.

$\mathcal{B}^{\mathsf{U}}$ perfectly simulates $\mathbf{G}_4$ when run with KGen-generated public keys, $\mathbf{G}_5$ when run with uniform public keys, and returns 1 if $\mathcal{A}$ wins the respective game. Since $\mathcal{B}^{\mathsf{U}}$ uses at most $n = q_{\texttt{IC.dec}} + q_{\mathsf{S}} + q_{\mathsf{E}}$ many public keys in total, the difference between $\mathcal{A}$'s winning probabilities in games $\mathbf{G}_4$ and $\mathbf{G}_5$ is upper bounded by the $n := q_{\texttt{IC.dec}} + q_{\mathsf{S}} + q_{\mathsf{E}}$-anonymity advantage of $\mathcal{B}^{\mathsf{U}}$ against KEM:

$$|\mathbf{Adv}_4 - \mathbf{Adv}_5| \leq \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\texttt{IC.dec}} + q_{\mathsf{S}} + q_{\mathsf{E}}) - \mathsf{PKU}}$$

5.3 Preparing to handle messages that involve the correct password

To quantify the protocol's leakage of the password, we will randomize protocol messages in section 5.4. For these randomizations to go unnoticed, we need to rule out the case that the adversary sent messages constructed from the correct password. (In many cases, the adversary would be able to derive some or all of the protocol outputs when using the correct password, thereby being able to notice the randomization.)

```
Adversary $\mathcal{B}^{\mathsf{U}}(\boldsymbol{pk})$
pkIndex = 0
$b \xleftarrow{unif} \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^b}(\boldsymbol{pk})$
output $b'_{\mathsf{PKU}} := [b = b']$
```

```
IC.dec($k_{pw}, apk$)
if $\exists$ record $(k_{pw}, pk, apk)$
    return $pk$
else
    $pk'_{\$} \leftarrow \boldsymbol{pk}$[pkIndex]                //$pk' \xleftarrow{unif} \mathcal{PK}$ vs
    $(pk'_{\$}, sk'_{\$}) \leftarrow\$$ KGen
    pkIndex += 1
    if $\exists$ record $(k_{pw}, pk'_{\$}, \star)$: corrPW
    if $\exists$ record $(\star, pk'_{\$}, apk)$: corrPW
    create record $(k_{pw}, pk'_{\$}, apk)$
    return $pk'$
```

Fig. 13: PKU adversary $\mathcal{B}^{\mathsf{U}}$, used to reason about the hop from game $\mathbf{G}_4$ to $\mathbf{G}_5$.

Therefore, whenever the adversary makes a `Send` query, we start to check if the correct password was used. (We will not need to do this check for Execute queries.) If such a query occurs, there are three possible reasons:

1. The adversary obtained the password by *corrupting* one of the parties involved in the session. In that case, we raise the trivial guess flag trivGuess for that session and continue the protocol without applying the changes of the following games, i.e., without randomisation. (Since the session is no longer fresh, it cannot be subjected to a `Test` query, meaning the adversary cannot using this session. Leakage of the password also is no longer a concern.)
2. The adversary is *forwarding* a message that was generated honestly by a previous flow of the `Send` oracle. Clearly, this event does not imply that the adversary has guessed the password of that session or knows any of the secret information associated with the message. Therefore, we do not count this as a correct guess and continue by randomizing the outputs according to section 5.4. This case is detected by keeping a record of all honestly generated transcripts.
3. The adversary *guessed* the password. We call this event `corrPW` and abort the game whenever it occurs. This way, no `Test` query can be issued to such a session, and we do not have to randomize the protocol messages. With the password already having been guessed, we no longer have to consider any further leakage about it. Throughout the games, we will bound how the probability of event `corrPW` changes, until we end up with a game in which we can bound the probability of event `corrPW` in terms of the dictionary size. (Even if the protocol reveals nothing on the password, an adversary can always try guessing.)

In conclusion, we will show that we can make the protocol messages independent of the password for all sessions where neither 1. nor 3. occurred.

Game $\mathbf{G}_6$*: Abort on Correct Password* A correct password guess is reflected in two cases, depending on which side the adversary sits:

Authenticated Public Key (apk) Consider the event where the adversary sent an apk built from the correct password *and* that the respective message was indeed generated by the adversary, i.e., that the message was not honestly generated by a previous call to `Send`. We'll

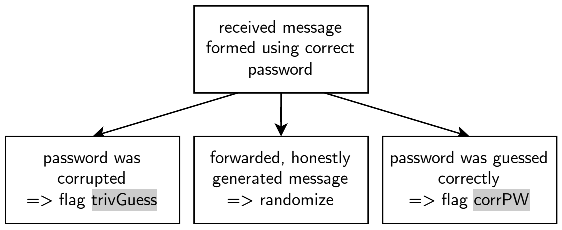
received message formed using correct password
password was corrupted => flag trivGuess
forwarded, honestly generated message => randomize
password was guessed correctly => flag corrPW

Fig. 14: The case distinction for messages formed using the correct password of a session.

call this event `apkCorrPw`. It can be detected as follows:
Whenever there is a query `Send`(apk^*), we look through the ideal cipher records for the record $(k_{pw}, pk, apk^*, "enc")$, where k_{pw} is the password-derived key for that session's password. Due to the changes in the previous two games, there cannot be a record for another password in this case. Therefore, if this record exists, we know that apk^* was generated using the correct password. If, additionally, apk was generated by the adversary (and not forwarded from another oracle), it contains a public key that was chosen by the adversary. If a game where this event occurs were to continue, the adversary should be able to decrypt all following messages in this session and receive the encapsulated key, thereby being able to distinguish the session key in the test query. [8]

Server Tag (tag_i) Consider the event where the adversary sent a tag_i built from the correct password *and* that the respective message was indeed generated by the adversary. We'll call this event `tagCorrPw`. It can be detected as follows:
When the adversary submits a responder tag to the `Send` oracle, we can look for records in `H` that link this tag to the password used to create it. By the changes made in previous games, it is ensured that there can be no collisions in `H` unless the game aborts, so there is at most one record for this tag, uniquely determining the *password* that was tested in this query. If the password in the record matches the correct one for this session, we conclude that the correct password was used to create this tag.

Combining the two cases and ruling out corruptions, we let `corrPW` be the event that either `apkCorrPw` or `tagCorrPw` occur *and* that neither party in the session was corrupted.

We let game $\mathbf{G}_6$ abort whenever `corrPW` occurs. Since both games proceed identically unless `corrPW` occurs, we have

$$|\mathbf{Adv}_5 - \mathbf{Adv}_6| \leq \Pr[\texttt{corrPW}_{\mathbf{G}_6}]$$

We will track how the probability of event `corrPW` changes throughout the sequence of games, and finish by bounding its probability in game 10.

5.4 Randomizing Protocol Messages (SG2)

Our next goal is to replace the protocol messages to make them independent of the password and the session key. We then argue that the modified game is indistinguishable to the adversary,

[8] Note that the existence of a record $(k_{pw}, pk, apk^*, "dec")$ is of no concern because it implies the adversary queried decryption of apk^* and received a public key pk that was chosen at random. In fact, the adversary can make any number of such queries and learn the possible public keys used to generate the response. We will argue in games 7 and 8 that this information is not helpful to the adversary.

using anonymity and indistinguishability of KEM. Using these computational assumptions, we can bound the amount of information that the protocol messages leak concerning the password.

As stated above, the adversary could notice this if they used the correct password to create a session. But this case can now be ruled out since the game then aborts, anyways. We only need to keep track of how the probability of `corrPW` changes, which we can also bound in terms of the computational assumptions on KEM since `corrPW` is an event that can be checked by a respective .

Game $\mathbf{G}_7$*: Randomize Encapsulation Public Key* In the first randomization step, we make the following change for all queries to the `Send` or `Execute` oracles where flag `trivGuess` is not raised: The public key used for the encapsulation is now generated independently of the password and the previously sent session messages, see the pseudo-code in figure 15.

We will now argue that an adversary noticing this change can be used to attack the multi-user anonymity property $(n, q_C) - \mathsf{ANO\text{-}CPA}$ where $n := q_{\mathtt{IC.dec}} + q_\mathsf{S} + q_\mathsf{E}$ and $q_C := q_\mathsf{S} + 1$. Intuitively, parameter n represents the number of public-keys in the reduction and is equal to the total number of potential public keys for any ciphertext c output by the `Send` or `Execute` oracles. Since the `Send` and `Execute` oracles query `IC.dec`, the number of sessions has to be added to the number of `IC.dec` queries the adversary is allowed to make.

Parameter q_C represents the maximal number of challenges issued for a given key pair, and is equal to the number of times an adversary could replay an authenticated public key, which is upper bounded by the total number of sessions.

We define adversary $\mathcal{B}_0^\mathsf{A}$ against the $(n, q_C) - \mathsf{ANO\text{-}CPA}$ experiment (defined in Fig. 3) as follows (for the sake of formality, we give the pseudo-code of $\mathcal{B}_0^\mathsf{A}$ in Fig. 16):

$\mathcal{B}_0^\mathsf{A}$ receives two vectors of challenge public keys $(\boldsymbol{pk_0}, \boldsymbol{pk_1})$ of dimension $n = q_{\mathtt{IC.dec}} + q_\mathsf{S} + q_\mathsf{E}$, and can query its challenge oracle `Chall`, provided by its $(n, q_C) - \mathsf{ANO\text{-}CPA}$ challenger, at most $q_C = q_\mathsf{S} + 1$ many times. (Depending on the challenger's bit, the challenges are generated using either $\boldsymbol{pk_0}$ or $\boldsymbol{pk_1}$.) $\mathcal{B}_0^\mathsf{A}$ samples an own challenge bit b', runs $\mathcal{A}$ and answers $\mathcal{A}$'s queries to the Oracles `H`, `IC.enc`, `KDF`, `Reveal`, and $\mathtt{Test}^{b'}$ according to the oracles in $\mathbf{G}_6$. When simulating ideal cipher decryption queries, $\mathcal{B}_0^\mathsf{A}$ embeds the challenge public keys contained in $\boldsymbol{pk_0}$ (see Fig. 16).

Whenever $\mathcal{A}$ queries the `Send`(.,.,.,flow=1) or `Execute` oracle, the ideal cipher decryption oracle is evaluated on the *apk* value sent by the initiator and the password-derived key of that session. Due to the abort conditions in game $\mathbf{G}_3$, we know that the record for *apk* and *pw* was not the result of a ideal cipher `encryption` query and the query returns one of the embedded challenge public keys. Therefore, there exists $j \in [n]$ s.t. $pk' = pk_{0,j}$.

To answer the query, $\mathcal{B}_0^\mathsf{A}$ then issues a query `Chall(`j`)` to receive a challenge (c^*, K^*), outputs c^* to $\mathcal{A}$ and uses K^* as K (see Fig. 16). (Ciphertexts returned by the `Send` or `Execute` oracle are therefore either encapsulations under the public key $pk_{0,j}$ or under $pk_{1,j}$, depending on the challenge bit in the $(n, q_C) - \mathsf{ANO\text{-}CPA}$ game. Note that since $\mathcal{A}$ can replay an *apk* value in each of the q_S many sessions, the same public key will sometimes be used to obtain multiple challenges. This is why we bound the number of challenges per key by $q_C = q_\mathsf{S} + 1$.)

When $\mathcal{A}$ outputs a guess b, $\mathcal{B}_0^\mathsf{A}$ checks if $b = b'$. In the case that `corrPW` did not occur and that $b = b'$, it returns 1 as its own output bit, otherwise, it returns 0.

$\mathcal{B}_0^\mathsf{A}$ perfectly simulates $\mathbf{G}_6$ when run with challenge bit 0, $\mathbf{G}_7$ when run with with challenge bit 1, and returns 1 if the adversary wins. Therefore, the difference between $\mathcal{A}$'s winning probabilities in games $\mathbf{G}_6$ and $\mathbf{G}_7$ is upper bounded by the respective $(n, q_C) - \mathsf{ANO\text{-}CPA}$ advantage of $\mathcal{B}_0^\mathsf{A}$ against KEM:

$$|\Pr[\mathbf{G}_6 \Rightarrow 1] - \Pr[\mathbf{G}_7 \Rightarrow 1]| \leq \mathbf{Adv}_{\mathsf{KEM}}^{(q_S + q_E, q_S + 1) - \mathsf{ANO\text{-}CPA}}(\mathcal{B}_0^{\mathsf{A}})$$

We note that we keep sessions involving corrupted parties (meaning `trivGuess` was raised for that session) unaffected of this change. As these sessions are by definitions `unfresh`, however, they cannot lead to a win for the adversary if tested.

To keep track of the change in the probability of Pr[`corrPW`], we can slightly adapt the reduction $\mathcal{B}_0^{\mathsf{A}}$: our new reduction $\mathcal{B}_1^{\mathsf{A}}$ behaves exactly like $\mathcal{B}_0^{\mathsf{A}}$ except how its output bit is defined. $\mathcal{B}_1^{\mathsf{A}}$ returns 1 if `corrPW` occured, and otherwise 0.

$$|\Pr[\texttt{corrPW}_{\mathbf{G}_6}] - \Pr[\texttt{corrPW}_{\mathbf{G}_7}]| \leq \mathbf{Adv}_{\mathsf{KEM}}^{(q_S + q_E, q_S + 1) - \mathsf{ANO\text{-}CPA}}(\mathcal{B}_1^{\mathsf{A}})$$

Oracle $\texttt{Send}_{\mathbf{G}_6}$`(P, i, msg, flow)` | $\texttt{Send}_{\mathbf{G}_7}$`(P, i, msg, flow)`

```
if $flow = 1$
    $apk \xleftarrow{\texttt{parse}} msg$
    $k_{pw} \leftarrow \texttt{KDF}(pw)$
    $pk' \leftarrow \texttt{IC.dec}(k_{pw}, apk)$




    $(c, K) \leftarrow\$ \mathsf{Encap}(pk')$
    $tag_i \leftarrow \texttt{H}(pw, apk, pk', c, K, "r")$
    return c, $tag_i$
```

```
if $flow = 1$
    $apk \xleftarrow{\texttt{parse}} msg$
    $k_{pw} \leftarrow \texttt{KDF}(pw)$
    if $\mathrm{PK}[(k_{pw}, apk)] \neq \perp$:
        $pk'_\$ \leftarrow \mathrm{PK}[(k_{pw}, apk)]$
    else
        $(pk'_\$, sk_\$) \leftarrow\$ \mathsf{KGen}$

    $\mathrm{PK}[(k_{pw}, apk)] \xleftarrow{\texttt{set}} pk'_\$$
    $(c, K) \leftarrow\$ \mathsf{Encap}(pk'_\$)$
    $tag_i \leftarrow \texttt{H}(pw, apk, pk'_\$, c, K, "r")$
    return c, $tag_i$
```

Fig. 15: Game $\mathbf{G}_6$: Randomizing public key in `Send` queries. This change in principle only needs to be done for queries where the *apk* value submitted by the adversary is not indicative of a correct guess (thereby triggering a game abort), according to the definition in subsection 5.3. The dictionary PK is a book-keeping tool introduced in game $\mathbf{G}_7$ to ensure that replays of the same value result in the same behavior. Note that this mirrors exactly what happens inside the ideal cipher.

With this change, the ciphertext c output by the `Send` and `Execute` oracles (for non-corrupted parties) is now independent of the password.

```
Adversary $\mathcal{B}_0^{\mathsf{A}}(\boldsymbol{pk_0}, \boldsymbol{pk_1})$
1 $\texttt{pkIndex} = 0$
2 $b \xleftarrow{unif} \{0,1\}$
3 $b' \leftarrow \mathcal{A}^{\mathcal{O}^b}()$
4 $b'_{\mathsf{ANO}} := [b = b']$
5 output $b'_{\mathsf{ANO}}$

$\texttt{IC.dec}(k_{pw}, apk)$
1 if $\exists$ record $(k_{pw}, pk, apk)$
2     return $pk$
3 else
4     $pk' \leftarrow \boldsymbol{pk_0}[\texttt{pkIndex}]$        $//pk' \leftarrow\!\!\$\ \mathsf{KGen}$
5     $\texttt{pkIndex}{+}{=}\ 1$
6     if $\exists$ record $(k_{pw}, pk', \star)$: abort
7     if $\exists$ record $(\star, pk', apk)$: abort
8     create record $(k_{pw}, pk', apk)$
9     return $pk'$

Oracle Send(P, i, msg, flow)
1 if $flow = 1$
2     $apk \xleftarrow{\texttt{parse}} msg$
3     $k_{pw} \leftarrow \texttt{KDF}(pw)$
4     $pk' \leftarrow \texttt{IC.dec}(k_{pw}, apk)$
5     find $j$ s.t. $pk' = \boldsymbol{pk_0}[j]$    //IC returned challenge $pk$ from
6     $\boldsymbol{pk_0}$
7     $(c, K) \leftarrow \texttt{Chall}(j)$        $//(c, K) \leftarrow \mathsf{Encap}(pk')$
8     $tag_i \leftarrow \texttt{H}(pw, apk, pk', c, K, "r")$
9     return c, $tag_i$
```

Fig. 16: $(n, q_C) - \mathsf{ANO\text{-}CPA}$ adversary $\mathcal{B}_0^{\mathsf{A}}$, used to reason about the hop from game $\mathbf{G}_6$ to $\mathbf{G}_7$. $\mathcal{O} = \{\texttt{KDF}, \texttt{IC.enc}, \texttt{IC.dec}, \texttt{Execute}, \texttt{Send}, \texttt{Reveal}, \texttt{Corrupt}\}$

Game $\mathbf{G}_8$*: Randomize Session Pre-Key* For all queries to the `Send` or `Execute` oracles where flag `trivGuess` is not raised, we now randomize the pre-key K that is used to derive the final session key and the responder tag, see the pseudo-code in figure 17. This change will make the pre-key independent of the ciphertext and the password for all sessions that are fresh. To keep the game consistent, however, we will need to let both sides of an honestly executed session use the same key. This is where correctness errors come into play since originally, both sides might have ended up with different keys in that case

We will now argue that an adversary noticing this change can be used to attack the indistinguishability property of the KEM. We define adversary $\mathcal{B}_0^{\mathsf{I}}$ against the $(n, q_C) - \mathsf{IND\text{-}CPA}$ experiment (defined in Fig. 2) as follows (for the sake of formality, we give the pseudo-code of $\mathcal{B}_0^{\mathsf{I}}$ in Fig. 18):

$\mathcal{B}_0^{\mathsf{I}}$ receives a vector of challenge public keys $\boldsymbol{pk}$, of dimension $n = q_{\mathsf{S}}$ and can query its challenge oracle `Chall`, provided by its $(n, q_C) - \mathsf{IND\text{-}CPA}$ challenger, at most $q_C = q_{\mathsf{S}} +$

```
Oracle Send$_{\mathbf{G}_7}$(P, i, msg, flow)                                        Send$_{\mathbf{G}_8}$(P, i, msg, flow)
if $flow = 1$                                                                        //first protocol message
  $apk \xleftarrow{\texttt{parse}} msg$
  $k_{pw} \leftarrow \texttt{KDF}(pw)$
  if $\mathrm{PK}[(k_{pw}, apk)] \neq \perp$:
    $pk'_{\$} \leftarrow \mathrm{PK}[(k_{pw}, apk)]$
  else:
    $(pk'_{\$}, sk_{\$}) \leftarrow\!\$\ \mathsf{KGen}$
    $\mathrm{PK}[(k_{pw}, apk)] \xleftarrow{\texttt{set}} pk'_{\$}$
  $(c, K) \leftarrow\!\$\ \mathsf{Encap}(pk'_{\$})$
                                                                                     $K_{\$} \xleftarrow{unif} \mathcal{K}$
  $tag_i \leftarrow \texttt{H}(pw, apk, pk'_{\$}, c, K, "r")$                        $tag_i \leftarrow \texttt{H}(pw, apk, pk'_{\$}, c, K_{\$}, "r")$
  return c, $tag_i$
if $flow = 2$                                                                        //second protocol message
  $c, tag_i \xleftarrow{\texttt{parse}} msg$
  $K' \leftarrow \mathsf{Decap}(sk, c)$
  if $tag_i = \texttt{H}(pw, apk, pk, c, K', "r")$    if $\exists$ record $tag_i = \texttt{H}(pw, apk, pk, c, K_{\$}, "r")$: $K'_{\$} \leftarrow K_{\$}$
    $SK \leftarrow \texttt{KDF}'(tag_i, K')$                                        $SK \leftarrow \texttt{KDF}'(tag_i, K'_{\$})$
    $\mathrm{K}[(\mathrm{P,i})] \xleftarrow{\texttt{set}} SK$
    $tag_r \leftarrow \texttt{H}(pw, apk, pk, c, K', "i")$                          $tag_r \leftarrow \texttt{H}(pw, apk, pk, c, K'_{\$}, "i")$
    return $tag_r$
  else return $\perp$
if $flow = 3$                                                                        //third protocol message
  $tag_r \xleftarrow{\texttt{parse}} msg$
  if $tag_r = \texttt{H}(pw, apk, pk', c, K, "i")$                                  if $tag_r = \texttt{H}(pw, apk, pk'_{\$}, c, K_{\$}, "i")$
    $SK \leftarrow \texttt{KDF}'(tag_r, K)$                                         $SK \leftarrow \texttt{KDF}'(tag_r, K_{\$})$
    $\mathrm{K}[(\mathrm{P,i})] \xleftarrow{\texttt{set}} SK$
```

Fig. 17: In game $\mathbf{G}_8$, the pre-key set after querying `Send` or `Execute` for some parties P_i, P_j will be independent of the password and the previous messages.

1 many times. (Depending on the challenger's bit, the challenge keys are either properly encapsulated keys or uniformly random.)

$\mathcal{B}_0^{\mathsf{I}}$ samples an own challenge bit b', runs $\mathcal{A}$ and answers $\mathcal{A}$'s queries to the Oracles H, IC.enc, KDF, Reveal, and $\mathtt{Test}^{b'}$ according to the oracles in $\mathbf{G}_7$.
(Contrary to the reduction in the previous game hop, $\mathcal{B}_0^{\mathsf{I}}$ does not have to embed the challenge public keys in the ideal cipher any more – due to the previous game hop, the public keys used for the encapsulation are now independent from the ones in the ideal cipher records.)

Now, whenever $\mathcal{A}$ queries the Send(.,.,.,.,1) or Execute oracle, $\mathcal{B}_0^{\mathsf{I}}$ will use one of the challenge public keys in $\boldsymbol{pk}$ (instead of generating a fresh public key), and issue a respective Chall query to its own challenger to receive a challenge (c^*, K^*), where j is the index of the respective public key. Upon receiving a challenge (c, K), $\mathcal{B}_0^{\mathsf{I}}$ uses it to answer the query.

If the same apk is submitted multiple times for sessions using the same password, the game is kept consistent by re-using the respective public key. (As discussed in the previous game hop, an adversary can replay an apk value $q_{\mathtt{S}}$ times, so the maximal number of queries to the challenge oracle for every key is $q_C = q_{\mathtt{S}} + 1$.)

Now, whenever $\mathcal{A}$ queries the Send(.,.,.,.,1) or Execute oracle, $\mathcal{B}_0^{\mathsf{I}}$ will look for a record matching the tag to use the correct challenge key.

When $\mathcal{A}$ outputs a guess b, $\mathcal{B}_0^{\mathsf{A}}$ checks if $b = b'$. In the case that corrPW did not occur and that $b = b'$, it returns 1 as its own output bit, otherwise, it returns 0.

$\mathcal{B}_0^{\mathsf{I}}$ perfectly simulates $\mathbf{G}_7$ when run with challenge bit 0 unless a correctness error occured for an honestly executed session, $\mathbf{G}_8$ when run with with challenge bit 1, and returns 1 if the adversary wins. Therefore, the difference between $\mathcal{A}$'s winning probabilities in games $\mathbf{G}_7$ and $\mathbf{G}_8$ is upper bounded by the respective $(q_{\mathtt{S}} + q_{\mathtt{E}}, q_{\mathtt{S}} + 1) - \mathsf{IND\text{-}CPA}$ advantage of $\mathcal{B}_0^{\mathsf{I}}$ against KEM:

$$|\mathbf{Adv}_7 - \mathbf{Adv}_8| \leq \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathtt{S}} + q_{\mathtt{E}}, q_{\mathtt{S}} + 1) - \mathsf{IND\text{-}CPA}}(\mathcal{B}_0^{\mathsf{I}}) + (q_{\mathtt{S}} + q_{\mathtt{E}}) \cdot \delta$$

To keep track of the change in the probability of Pr[corrPW], we can adapt the reduction $\mathcal{B}_0^{\mathsf{I}}$ exactly like in the game-hop before by redefining the output bit to be 1 iff corrPW occured.

$$|\Pr[\mathtt{corrPW}_{\mathbf{G}_7}] - \Pr[\mathtt{corrPW}_{\mathbf{G}_8}]| \leq \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathtt{S}} + q_{\mathtt{E}}, q_{\mathtt{S}} + 1) - \mathsf{IND\text{-}CPA}}(\mathcal{B}_1^{\mathsf{I}})$$

We again note that while sessions involving corrupted parties are also unaffected by this change, these sessions are by definitions unfresh and cannot lead to a win for the adversary if tested.

At this point, the pre-key K (for sessions between non-corrupted parties) is independent of both the password and the protocol messages.

Game $\mathbf{G}_9$*: Randomize Server Tag* To argue that the responder tag does not leak significant information on the password or the session key, we replace it with a random value. The change for Send queries is shown in figure 19.

Let TagQueried be the event that the adversary has queried $\mathtt{H}(pw, apk, pk'_{\$}, c, K_{\$}, "r")$. We argue that due to the properties of the random oracle H, games $\mathbf{G}_8$ and $\mathbf{G}_9$ are indistinguishable to the adversary unless TagQueried occurs. Since the pre-key K is independent of the attacker's view the best attack is random guessing. Games $\mathbf{G}_8$ and $\mathbf{G}_9$ are identical unless TagQueried occurs. Therefore, if $\mathcal{A}$ can issue at most $q_{\mathtt{H}}$ queries to the random oracle H, we have

$$|\mathbf{Adv}_8 - \mathbf{Adv}_9|, |\Pr[\mathtt{corrPW}_{\mathbf{G}_8}] - \Pr[\mathtt{corrPW}_{\mathbf{G}_9}]| \leq \Pr[\mathtt{TagQueried}] \leq \frac{q_{\mathtt{H}} \cdot (q_{\mathtt{S}} + q_{\mathtt{E}})}{|\mathcal{K}|}$$

```
Adversary $\mathcal{B}_0^{\mathsf{I}}(\boldsymbol{pk})$
pkIndex = 0
$b \xleftarrow{unif} \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^b}(\boldsymbol{pk})$
$b'_{\mathsf{IND}} := [b = b']$
output $b'_{\mathsf{IND}}$
```

```
Oracle Send(P, i, msg, flow)
if $flow = 1$                                          //first protocol message
  $apk \xleftarrow{\texttt{parse}} msg$
  $k_{pw} \leftarrow \texttt{KDF}(pw)$
  if $\exists$ record $\mathrm{PK}[(k_{pw}, apk)]$:
    $pk'_{\$} \leftarrow \mathrm{PK}[(k_{pw}, apk)]$
  else:
    $pk'_{\$} \leftarrow \boldsymbol{pk}[\texttt{pkIndex}]$
    pkIndex += 1
    $\mathrm{PK}[(k_{pw}, apk)] \xleftarrow{\texttt{set}} pk'_{\$}$
  find $j$ s.t. $pk'_{\$} = \boldsymbol{pk}_j$            // may not be unique but that's OK
  $(K^*, c^*) \leftarrow$ Chall(j)                       // $K^*$ is real or random w.r.t. $c^*$
  $tag_i \leftarrow \texttt{H}(pw, apk, pk'_{\$}, c^*, K^*, "r")$
  return c, $tag_i$
if $flow = 2$                                          //second protocol message
  $c, tag_i \xleftarrow{\texttt{parse}} msg$
  if $\exists$ record $tag_i = \texttt{H}(pw, apk, pk, c, K_{\$}, "r")$:
    $K'_{\$} \leftarrow K_{\$}$
    $SK \leftarrow \texttt{KDF}'(tag_i, K'_{\$})$             //insert challenge $K'_{\$} = K^*$
    $\mathrm{K[(P,i)]} \xleftarrow{\texttt{set}} SK$
    $tag_r \leftarrow \texttt{H}(pw, apk, pk, c, K'_{\$}, "i")$
    return $tag_r$
  else return $\perp$
```

Fig. 18: (n, q_C) − IND-CPA adversary $\mathcal{B}_0^{\mathsf{I}}$, used to reason about the hop from game $\mathbf{G}_7$ to $\mathbf{G}_8$. $\mathcal{O} = \{\texttt{KDF}, \texttt{IC.enc}, \texttt{IC.dec}, \texttt{Execute}, \texttt{Send}, \texttt{Reveal}, \texttt{Corrupt}\}$

```
Oracle $\mathtt{Send}_{\mathbf{G}_8}(P, i, msg, flow)$                          $\mathtt{Send}_{\mathbf{G}_9}(P, i, msg, flow)$
if $flow = 1$ //first protocol message
    $apk \xleftarrow{\mathtt{parse}} msg$
    $k_{pw} \leftarrow \mathtt{KDF}(pw)$
    $(pk'_{\$}, sk) \leftarrow\!\$\ \mathsf{KGen}$
    $(c, K) \leftarrow\!\$\ \mathsf{Encap}(pk'_{\$})$
    $K_{\$} \xleftarrow{unif} \mathcal{K}$
    $tag_i \leftarrow \mathtt{H}(pw, apk, pk'_{\$}, c, K_{\$}, "r")$              $tag_{i\$} \xleftarrow{unif} \mathcal{T}$
    return c, $tag_i$                                                 return c, $tag_{i\$}$
```

Fig. 19: Randomizing tags in `Send` queries. $\mathcal{T}$ is the domain of the random oracle `H`.

5.5 Randomizing Session Key (SG3)

Game $\mathbf{G}_{10}$*: Randomize Session Key* Finally, we replace the final session key for all `Send` and `Execute` queries with one chosen independently at random from the session key space $\mathcal{SK}$, meaning they are now independent of the previous messages and the password. This change is only done for sessions where flag `trivGuess` did not occur.

Let `SKQueried` be the event that the adversary has queried $\mathtt{KDF}(tag_i, K_{\$})$. In game $\mathbf{G}_{10}$, we abort whenever this occurs. We argue that due to the properties of the random oracle `KDF`', games $\mathbf{G}_9$ and $\mathbf{G}_{10}$ are indistinguishable to the adversary unless `SKQueried` occurs. Since each pre-key $K_{\$}$ is independent of the attacker's view the best attack is random guessing. Therefore, for an adversary that can issue at most $q'_{\mathtt{KDF}}$ queries to the random oracle `KDF`' and $q_{\mathtt{S}} + q_{\mathtt{E}}$ potential session keys, we can bound the probability of this event. Since games $\mathbf{G}_9$ and $\mathbf{G}_{10}$ are identical unless `SKQueried` occurs, we have that

$$|\mathbf{Adv}_9 - \mathbf{Adv}_{10}|, |\Pr[\mathtt{corrPW}_{\mathbf{G}_9}] - \Pr[\mathtt{corrPW}_{\mathbf{G}_{10}}]| \leq \Pr[\mathtt{SKQueried}] \leq \frac{q_{\mathtt{KDF}'} \cdot (q_{\mathtt{S}} + q_{\mathtt{E}})}{|\mathcal{SK}|}$$

After this change, the adversary has to test a session that will always respond with a uniformly random value that is independent of the challenge bit. We can conclude that the winning probability of $\mathcal{A}$ in game $\mathbf{G}_{10}$ is therefore reduced to that of random guessing:

$$\mathbf{Adv}_{10} = \frac{1}{2}.$$

Bounding Correct Password Event All of the responder's protocol messages are now independent of the respective password for all sessions that are fresh, meaning they do not give the adversary any information about the passwords. However, the adversary can still attempt a password guess by picking a password from the password space and using it in a `Send` query. We can bound the probability of a correct random guess depending on the number of send queries and the distribution of the passwords. Assuming a uniform distribution on a password dictionary of size $|\mathcal{D}|$, and assuming $\mathcal{A}$ issues $q_{\mathtt{S}}$ many send queries, we get the bound

$$\Pr[\mathtt{corrPW}_{\mathbf{G}_{10}}] \leq \frac{q_{\mathtt{S}}}{|\mathcal{D}|}.$$

```
Oracle $\mathtt{Send}_{\mathbf{G}_9}(P, i, msg, flow)$                                    $\mathtt{Send}_{\mathbf{G}_{10}}(P, i, msg, flow)$
if $\underline{flow = 2}$                                                               //second protocol message
  $c, tag_i \xleftarrow{\mathtt{parse}} msg$
  if $\exists$ record $tag_i = \mathtt{H}(pw, apk, pk, c, K_\$, "r")$: $K'_\$ \leftarrow K_\$$
    $SK \leftarrow \mathtt{KDF}'(tag_i, K'_\$)$                                           $SK_\$ \xleftarrow{unif} \mathcal{SK}$
    $\mathrm{K[(P,i)]} \xleftarrow{\mathtt{set}} SK$                                     $\mathrm{K[(P,i)]} \xleftarrow{\mathtt{set}} SK_\$$
    $tag_r \leftarrow \mathtt{H}(pw, apk, pk, c, K'_\$, "i")$
    **return** $tag_r$
  else **return** $\perp$
if $\underline{flow = 3}$                                                               //third protocol message
  $tag_r \xleftarrow{\mathtt{parse}} msg$
  $K' \leftarrow \mathsf{Decap}(sk, c)$
  if $tag_r = \mathtt{H}(pw, apk, pk'_\$, c, K_\$, "r")$ for some $K_\$$
                                                                                        if $K_\$ \neq K'$
    $SK \leftarrow \mathtt{KDF}'(tag_i, K_\$)$                                            $SK_\$ \leftarrow \mathrm{K}[(P, i)]$
    $\mathrm{K[(P,i)]} \xleftarrow{\mathtt{set}} SK$                                     $\mathrm{K[(P,i)]} \xleftarrow{\mathtt{set}} SK_\$$
```

Fig. 20: In game $\mathbf{G}_{10}$, the final session key is randomized.

Collecting the probabilities, we can now bound the probability of event $\texttt{corrPW}$ occurring in game 6:

$$\begin{aligned}
\Pr[\texttt{corrPW}_{\mathbf{G}_6}] &\leq \sum_{i=6}^{9}(\Pr[\texttt{corrPW}_{\mathbf{G}_i}] - \Pr[\texttt{corrPW}_{\mathbf{G}_{i+1}}]) + \Pr[\texttt{corrPW}_{\mathbf{G}_{10}}] \\
&\leq \frac{q_{\mathsf{S}}}{|\mathcal{D}|} + \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathsf{S}} + q_{\mathtt{E}}, q_{\mathsf{S}} + 1) - \mathsf{ANO\text{-}CPA}}(\mathcal{B}_1^{\mathsf{A}}) \\
&\quad + \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathsf{S}} + q_{\mathtt{E}}, q_{\mathsf{S}} + 1) - \mathsf{IND\text{-}CPA}}(\mathcal{B}_1^{\mathsf{I}}) + \frac{q_{\mathtt{H}} \cdot (q_{\mathsf{S}} + q_{\mathtt{E}})}{|\mathcal{K}|} + \frac{q_{\mathtt{KDF}'} \cdot (q_{\mathsf{S}} + q_{\mathtt{E}})}{|\mathcal{SK}|}
\end{aligned}$$

5.6 Total bound on BPR security

To wrap up the proof, we can now bound the BPR advantage of an adversary against the PAKEM protocol, using the triangle inequality. We will also fold the two anonymity adversaries $\mathcal{B}_0^{\mathsf{A}}$ and $\mathcal{B}_1^{\mathsf{A}}$ into one ($\mathcal{B}^{\mathsf{A}}$), as well as the two indistinguishability adversaries $\mathcal{B}_0^{\mathsf{I}}$ and $\mathcal{B}_1^{\mathsf{I}}$ ($\mathcal{B}^{\mathsf{I}}$).

$$\begin{aligned}
\mathbf{Adv}_{\mathsf{PAKEM}}^{\mathsf{BPR}}(\mathcal{A}) &\leq |\mathbf{Adv}_0 - \mathbf{Adv}_9| - \frac{1}{2} \\
&\leq \sum_{i=1}^{9}|\mathbf{Adv}_{\mathsf{i}} - \mathbf{Adv}_{\mathsf{i+1}}| - \frac{1}{2} \\
&= \frac{q_{\mathtt{KDF}}^2}{2 \cdot |\mathcal{K}_{pw}|} + \frac{q_{\mathtt{IC}}^2}{2 \cdot |\mathcal{PK}|} + \frac{q_{\mathtt{IC}}^2}{|\mathcal{PK}|} + \frac{q_{\mathtt{H}}^2}{2 \cdot |T|} + \underbrace{\Pr[\texttt{apkCorrPw}] + \Pr[\texttt{tagCorrPw}]}_{=\Pr[\texttt{corrPW}]} + \\
&\quad + \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathtt{IC.dec}} + q_{\mathsf{S}} + q_{\mathtt{E}}) - \mathsf{PKU}} + \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathsf{S}} + q_{\mathtt{E}}, q_{\mathsf{S}} + 1) - \mathsf{ANO\text{-}CPA}} + \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathsf{S}} + q_{\mathtt{E}}, q_{\mathsf{S}} + 1) - \mathsf{IND\text{-}CPA}} \\
&\quad + \frac{q_{\mathtt{H}} \cdot (q_{\mathsf{S}} + q_{\mathtt{E}})}{|\mathcal{K}|} + \frac{q_{\mathtt{KDF}'} \cdot (q_{\mathsf{S}} + q_{\mathtt{E}})}{|\mathcal{SK}|} + (q_{\mathsf{S}} + q_{\mathtt{E}}) \cdot \delta + \frac{1}{2} - \frac{1}{2} \\
&\leq \frac{q_{\mathsf{S}}}{|\mathcal{D}|} + \frac{q_{\mathtt{KDF}}^2}{2 \cdot |\mathcal{K}_{pw}|} + \frac{q_{\mathtt{IC}}^2}{2 \cdot |\mathcal{PK}|} + \frac{q_{\mathtt{IC}}^2}{|\mathcal{PK}|} + \frac{q_{\mathtt{H}}^2}{2 \cdot |T|} + \frac{q_{\mathtt{H}} \cdot (q_{\mathsf{S}} + q_{\mathtt{E}})}{|\mathcal{K}|} + \frac{q_{\mathtt{KDF}'} \cdot (q_{\mathsf{S}} + q_{\mathtt{E}})}{|\mathcal{SK}|} \\
&\quad + \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathtt{IC.dec}} + q_{\mathsf{S}} + q_{\mathtt{E}}) - \mathsf{PKU}}(\mathcal{B}^{\mathsf{U}}) + \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathsf{S}} + q_{\mathtt{E}}, q_{\mathsf{S}} + 1) - \mathsf{ANO\text{-}CPA}}(\mathcal{B}_0^{\mathsf{A}}) + \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathsf{S}} + q_{\mathtt{E}}, q_{\mathsf{S}} + 1) - \mathsf{ANO\text{-}CPA}}(\mathcal{B}_1^{\mathsf{A}}) \\
&\quad + \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathsf{S}} + q_{\mathtt{E}}, q_{\mathsf{S}} + 1) - \mathsf{IND\text{-}CPA}}(\mathcal{B}_0^{\mathsf{I}}) + \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathsf{S}} + q_{\mathtt{E}}, q_{\mathsf{S}} + 1) - \mathsf{IND\text{-}CPA}}(\mathcal{B}_1^{\mathsf{I}}) + (q_{\mathsf{S}} + q_{\mathtt{E}}) \cdot \delta \\
&\leq \frac{q_{\mathsf{S}}}{|\mathcal{D}|} + \mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathtt{IC.dec}} + q_{\mathsf{S}} + q_{\mathtt{E}}) - \mathsf{PKU}}(\mathcal{B}^{\mathsf{U}}) + 2\mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathsf{S}} + q_{\mathtt{E}}, q_{\mathsf{S}} + 1) - \mathsf{ANO\text{-}CPA}}(\mathcal{B}^{\mathsf{A}}) \\
&\quad + 2\mathbf{Adv}_{\mathsf{KEM}}^{(q_{\mathsf{S}} + q_{\mathtt{E}}, q_{\mathsf{S}} + 1) - \mathsf{IND\text{-}CPA}}(\mathcal{B}^{\mathsf{I}}) + \frac{3 \cdot q_{\mathtt{IC}}^2 \cdot}{2 \cdot |\mathcal{PK}|} + (q_{\mathsf{S}} + q_{\mathtt{E}}) \cdot \delta \\
&\quad + q_{\mathtt{RO}} \cdot (q_{\mathsf{S}} + q_{\mathtt{E}}) \cdot (\frac{1}{|\mathcal{SK}|} + \frac{1}{|\mathcal{K}|}) + q_{\mathtt{RO}}^2 \cdot (\frac{1}{2 \cdot |T|} + \frac{1}{2 \cdot |\mathcal{K}_{pw}|})
\end{aligned}$$

Notes on Reveal and Corrupt queries Under the restrictions outlined in the freshness condition (definition 8), an adversary can also obtain session keys by using `Reveal` queries and passwords using `Corrupt` queries. We argue that the proof given above holds in the presence of these queries.

Reveal: an adversary can reveal the session key of an instance that has terminated. Then this instance and the instance running the other side of the protocol (if it exists) can no longer be subject to a `Test` query. However, if the protocol leaked information on the *password* if the session key is known, this would break security. To give an example of how password leakage could occur, imagine the following *modified* insecure protocol: the protocol is the same as the PAKEM protocol, but the final session key SK is not the output of KDF'(tag_i, K), but K. Then, revealing the session key K would give the adversary an oracle for the password: by computing candidate tag_i (or tag_r) values based on different values of pw, you can check which password gives the correct tag because the password space is small enough that a brute-force attack is feasible.

In the case where the final session key is derived from the pre-key K and the transcript, this attack on the password is *infeasible* since you would first need to brute-forcing the pre-key K from the identity $SK \leftarrow \texttt{KDF}'(tag_i, K)$ knowing only SK and tag_i. In game 10, the session key is randomized for all sessions that where the password was not guessed, demonstrating that SK does not reveal significant information on the password.

Corrupt: an adversary can reveal the passwords of a party. In that case, no session key derived from a session this party is involved in can be subject to a `Test` query. However, the adversary's view of the experiment must stay consistent throughout the game hops, even if this additional information is known to the adversary. An adversary that knows the password of a pair of parties could potentially notice that some protocol messages are at some point no longer derived from the password, which would mean that the view has changed. This would present a problem even if the adversary is forbidden from issuing a `Test` query on the instances of these parties, however we only randomize values dependent on the password for sessions where the password is not available to the adversary. This is accomplished in game 6, where we abort all experiments where a message was submitted that the adversary has created using the correct password of a non-corrupted session.

Because we want to achieve forward-secrecy, we also allow `Test` queries on corrupted parties, as long as the session that is tested is not subject to any `Send` queries. Therefore, we focus on the parts of the proof that affect `Execute` queries and see that by the end of the proof, all transcripts output by that oracle have a session key that is randomized. In fact, this follows directly from the IND-CPA property of KEM, which can be seen in game 8.

References

ABR$^+$21. Michel Abdalla, Manuel Barbosa, Peter B. Rønne, Peter Y. A. Ryan, and Petra Šala. Security characterization of j-pake and its variants. Cryptology ePrint Archive, Paper 2021/824, 2021. `https://eprint.iacr.org/2021/824`.

AFP05. Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 65–84, Les Diablerets, Switzerland, January 23–26, 2005. Springer, Heidelberg, Germany.

AHH21. Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of CPace. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part IV*, volume 13093 of *Lecture Notes in Computer Science*, pages 711–741, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany.

AHH23. Michel Abdalla, Björn Haase, and Julia Hesse. CPace, a balanced composable PAKE. Internet-Draft draft-irtf-cfrg-cpace-08, Internet Engineering Task Force, July 2023. Work in Progress.

BBC$^+$13. Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New techniques for SPHFs and efficient one-round PAKE protocols. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 449–475, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

BBDP01. Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 566–582, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany.

BBM00. Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 259–274, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.

BCJ$^+$19. Tatiana Bradley, Jan Camenisch, Stanislaw Jarecki, Anja Lehmann, Gregory Neven, and Jiayu Xu. Password-authenticated public-key encryption. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19: 17th International Conference on Applied Cryptography and Network Security*, volume 11464 of *Lecture Notes in Computer Science*, pages 442–462, Bogota, Colombia, June 5–7, 2019. Springer, Heidelberg, Germany.

BCP$^+$23. Hugo Beguinet, Céline Chevalier, David Pointcheval, Thomas Ricosset, and Mélissa Rossi. Get a cake: Generic transformations from key encapsulation mechanisms to password authenticated key exchanges. *Cryptology ePrint Archive*, 2023.

BDF$^+$11. Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 41–69, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

BDK$^+$18. Joppe Bos, Leo Ducas, Eike Kiltz, T Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM. In *IEEE (EuroS&P) 2018*, pages 353–367, 2018.

BFK09. Jens Bender, Marc Fischlin, and Dennis Kügler. Security analysis of the PACE key-agreement protocol. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *ISC 2009: 12th International Conference on Information Security*, volume 5735 of *Lecture Notes in Computer Science*, pages 33–48, Pisa, Italy, September 7–9, 2009. Springer, Heidelberg, Germany.

BHK$^+$19. Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ signature framework. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2129–2146, London, UK, November 11–15, 2019. ACM Press.

Bla06. John Black. The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In Matthew J. B. Robshaw, editor, *Fast Software Encryption – FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 328–340, Graz, Austria, March 15–17, 2006. Springer, Heidelberg, Germany.

BM92. Steven Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secureagainst dictionary attacks. *Security and Privacy, IEEE Symposium on*, 0:72, 04 1992.

BPR00. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

CFHL21. Kai-Min Chung, Serge Fehr, Yu-Hsuan Huang, and Tai-Ning Liao. On the compressed-oracle technique, and post-quantum security of proofs of sequential work. In Anne Canteaut and

François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 598–629, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

CHK$^+$05. Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 404–421, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

CLM$^+$18. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

Cou06. Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. `https://eprint.iacr.org/2006/291`.

DFMS21. Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Online-extractability in the quantum random-oracle model. Cryptology ePrint Archive, Report 2021/280, 2021. `https://eprint.iacr.org/2021/280`, accepted for publication at Eurocrypt 2022.

DFMS22. Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Online-extractability in the quantum random-oracle model. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 677–706, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany.

DHK$^+$21. Julien Duman, Kathrin Hövelmanns, Eike Kiltz, Vadim Lyubashevsky, and Gregor Seiler. Faster lattice-based KEMs via a generic fujisaki-okamoto transform using prefix hashing. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 2722–2737, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

DKL$^+$18. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):238–268, 2018. `https://tches.iacr.org/index.php/TCHES/article/view/839`.

DKS18. Luca De Feo, Jean Kieffer, and Benjamin Smith. Towards practical key exchange from ordinary isogeny graphs. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 365–394, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

GHHM21. Alex B. Grilo, Kathrin Hövelmanns, Andreas Hülsing, and Christian Majenz. Tight adaptive reprogramming in the QROM. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 637–667, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany.

GKP18. Federico Giacon, Eike Kiltz, and Bertram Poettering. Hybrid encryption in a multi-user setting, revisited. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 159–189, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Heidelberg, Germany.

GMP22. Paul Grubbs, Varun Maram, and Kenneth G. Paterson. Anonymous, robust post-quantum public key encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 402–432, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany.

HHM22. Kathrin Hövelmanns, Andreas Hülsing, and Christian Majenz. Failing gracefully: Decryption failures and the fujisaki-okamoto transform. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part IV*, volume 13794 of *Lecture Notes in*

Computer Science, pages 414–443, Taipei, Taiwan, December 5–9, 2022. Springer, Heidelberg, Germany.

HvO22. Feng Hao and Paul C. van Oorschot. Sok: Password-authenticated key exchange – theory, practice, standardization and real-world lessons. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, page 697–711, New York, NY, USA, 2022. Association for Computing Machinery.

HY18. Akinori Hosoyamada and Kan Yasuda. Building quantum-one-way functions from block ciphers: Davies-Meyer and Merkle-Damgård constructions. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 275–304, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

JKX18. Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 456–486, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

Lan16. Jean Lancrenon. On password-authenticated key exchange security modeling. In Frank Stajano, Stig F. Mjølsnes, Graeme Jenkinson, and Per Thorsheim, editors, *Technology and Practice of Passwords*, pages 120–143, Cham, 2016. Springer International Publishing.

PFH$^+$22. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`.

RS06. Alexander Rostovtsev and Anton Stolbunov. Public-Key Cryptosystem Based On Isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. `https://eprint.iacr.org/2006/145`.

Sha49. Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

Sho20. Victor Shoup. Security analysis of spake2+. Cryptology ePrint Archive, Paper 2020/313, 2020. `https://eprint.iacr.org/2020/313`.

Son14. Fang Song. A note on quantum security for post-quantum cryptography. In Michele Mosca, editor, *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014*, pages 246–265, Waterloo, Ontario, Canada, October 1–3, 2014. Springer, Heidelberg, Germany.

Zha19. Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 239–268, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.