





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

Nouri Alnahawi¹ * , Kathrin Hövelmanns² , Andreas Hülsing² ** , and Silvia Ritsch² *** 

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 common for PAKE. So far, however, it is not known how to model or prove computational UC security against quantum adversaries let alone if the proof uses 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, assuming the underlying KEM satisfies notions of ciphertext indistinguishability, anonymity, and (computational) public-key uniformity. To achieve tight security bounds, we use multi-user variants of the aforementioned properties.

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

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, which usually uses public-key cryptography 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 maintain 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 human-memorable passwords within the already established communication that is secured via the shared secret. This is the case as passwords are far

* 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 Skłodowska-Curie Action (PROJECT 101072637 - HORIZON - MSCA-2021-DN-01)

easier to handle by humans. However, this means that 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 user password for the purpose of authentication in an AKE. This is 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 requirements³ 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 a symmetric key in encrypted form. This allows the encapsulation algorithm to internally chose the key, instead of taking it as an input, and then return it 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 (NIKE) 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 ones, replacing quantum-vulnerable building blocks by the available KEM and signatures. A general challenge – which also concerns PAKE – is that the NIST proposals cannot replace the Diffie-Hellman key exchange in a ‘plug-n-play’ way: most PAKE protocols rely on the additional 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

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

- is versatile, i.e., designed in a way that works for various PQC proposals or even pre- and post-quantum hybrids (instead of being tied to a specific proposal’s internal workings);
- works with already proposed algorithms (to avoid having 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. Additionally, this also allows for the possibility to select 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 that considers quantum attacks.

At the same time, there is continuous progress in lifting game-based security results to a setting with quantum adversaries. 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 becomes less straight-forward. This is also the case for PAKEs. Both the ROM and the IC model do not account for quantum attacks and therefore make it necessary to adapt both the models and the proofs. By now, we have a somewhat well-understood quantum counterpart to the ROM, called quantum-accessible ROM (QROM) [BDF⁺11]. Ongoing efforts to develop the necessary techniques for lifting proofs to the QROM 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]. What is still missing, however, are detailed formal proofs for PAKE protocols that could be lifted.

Our contribution. In this work, we progress towards a PAKE protocol with proven security against quantum adversaries. Towards this end, we analyse 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 in which this step is hidden within 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 are not necessary for a game-

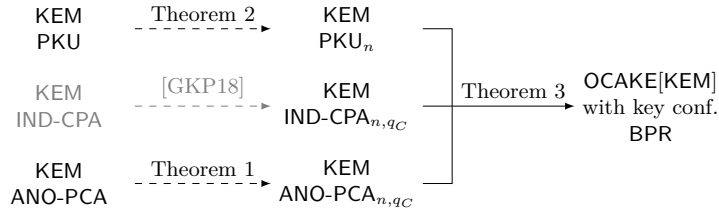


Fig. 1: Our results. Solid (dashed) arrows indicate tight (non-tight) reductions.

based proof. Second, we consciously add a final key confirmation message that achieves 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. (BPR already discussed 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 for future work in which either new techniques to lift results with ideal ciphers are developed, or the use of ideal ciphers is omitted.

Organization of this paper. After recalling basic notions (including the relevant security notions for KEMs) in Section 2 and introducing multi-user notions for KEMs in Section 3, we describe the OCAKE protocol extended with a key confirmation in Section 4. We recall the BPR security model for PAKE in Section 5, and then prove OCAKE secure in Section 6.

1.1 Concurrent work

A recent publication [PZ23] gave a game-based proof for CAKE, another protocol from the [BCP⁺23] PAKE family. As a nice side effect, availability of several proofs establishes trust in the soundness of the family’s overall design approach. To compare with [PZ23], we point out the main advantages of this work below.

Protocol advantages. Like OCAKE, our modification avoids the usage of ideal ciphers (ICs) for the second protocol message. When comparing to CAKE, it thus reduces the computational overhead as well as the conceptual overall involvement of ICs. So far, IC handling still poses a major barrier when proving security against quantum attackers, we thus followed the maxim ‘the less IC, the better’. (It cannot be ruled out, however, that involving ICs at all already hinders a proof against quantum attacks.)

As stated in [PZ23], CAKE was chosen over OCAKE because OCAKE only was known to achieve weak forward secrecy. This limitation seems to stem from the comparably weak anonymity requirement made in [BCP⁺23], and is overcome by strengthening the requirement in a way such that it still is achieved by – amongst others – the post-quantum KEMs Kyber [MX23], McEliece, NTRU, BIKE and SIKE (all [Xag22]), as well as FrodoKEM [GMP22].

KEM requirement advantages. Both works require anonymity and indistinguishability notions. In [PZ23], both notions involve an additional oracle (called PC0) that is accessible to the attacker. We improve this in two separate ways: 1.) Our indistinguishability notion works without involving the additional PC0 attack surface, thus leading to an easier-to-analyze (more standard) requirement

with potential for more efficient instantiations. 2.) We also attenuate the anonymity requirement (thus creating room for efficiency improvements):

- In [PZ23], the number of PCO queries might be high – it’s bounded by how many random oracle queries an attacker could reasonably perform. We only need to allow a single query. Only having to allow a single oracle query might allow simpler and more efficient designs (see, e.g., [HV22]).
- Adapting [PZ23] to the QROM would need to allow quantum access to PCO. This work would still only need to allow classical access, which simplifies the analysis and the KEM design. It is likely that quantum access to PCO limits with which parameters KEM can still be instantiated securely.

Recently, there were discussions [Ber22] concerning multi-user security of lattice-based cryptosystems (such as Kyber). As a minor addition, we thus enable a result that only requires single-user security by including (generic and non-tight) single-to-multi-user results.

Proof advantages. Both proofs deal with attackers that correctly guess a password by defining a respective ‘bad’ event in the security game and proving its probability to be small.

Our probability term compares favorably to the one in [PZ23]: the term in [PZ23] involves the number of *all* established sessions, including observed honest protocol runs (which will be very large in practice), whereas ours only involves the number of sessions with which an attacker can actively interfere (which in practice will be limited). We stress, however, that the term in [PZ23] can easily be shrunken down to our term with a more fine-grained analysis.

Additionally, our treatment of the ‘bad’ event might be easier to verify: In [PZ23], the event is treated by raising an internal flag and performing flag-dependent changes to the game. We early on change the game such that correct guesses are ‘punished’ by aborting and analyse how this probability is affected by subsequent game modifications. Second, the recognition of the ‘bad’ event differs due to protocol differences. CAKE encrypts both protocol messages via the ideal cipher, [PZ23] can thus identify password guesses by connecting the protocol message to a previous ideal cipher query. In our modification of OCAKE, the second message is not IC-encrypted. We can, however, identify password guesses via the included authentication tag since it is computed using a hash function (modelled as a random oracle).

The protocol differences also affects how the simulation deals with server impersonation. Since an attacker might corrupt a server password during a protocol run, the simulated client must be able to respond to ciphertexts generated by an attacker in possession of the public key. [PZ23] cover this case by simulating the involved random oracle in a certain way (‘oracle patching’). In particular, the random oracle internally calls the plaintext checking oracle upon each query. Our authentication tag is the reason why we do not need a random oracle patching technique- we can directly identify a password guess by connecting the tag to a previous random oracle query.

This allows us to a) limit the number of PCO queries to 1, and b) work with a non-quantum version of PCO even when adapting the proof to the QROM.

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

In the following we recall the ideal cipher model and provide definitions for key encapsulation mechanisms (KEM). We assume the reader is familiar with the random oracle model (ROM) [BR93].

The Ideal Cipher Model. We prove security of the OCAKE protocol in the ideal cipher model [Sha49],[Bla06]. Analogously to the ROM 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 key length k consists of two algorithms $BC.\text{enc} : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ and $BC.\text{dec} : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ such that for every plaintext $m \in \{0,1\}^n$ and key $k \in \{0,1\}^k$, decryption undoes encryption: $IC.\text{dec}(k, IC.\text{enc}(k, m)) = m$.

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 $IC.\text{enc}, IC.\text{dec} : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{M}$, such that $IC.\text{dec}(k, IC.\text{enc}(k, m)) = m$ and $IC.\text{enc}(k, IC.\text{dec}(k, m)) = m$ for all k, m in $\mathcal{K} \times \mathcal{M}$.

Key Encapsulation Mechanisms (KEM). We start with the functional definition of KEMs. Afterwards we discuss their security.

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

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

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

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

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

We use three security notions for KEMs: ANonymity under Plaintext-Checking Attacks (ANO-PCA), an extension of the anonymity notion given in [BBDP01, GMP22], INDistinguishability under Chosen-Plaintext-Attacks (IND-CPA), and Public Key Uniformity. We begin with the first two as they share the plaintext checking oracle (PCO). The presence of a PCO might look artificial at a first glance. Looking ahead, we will need it in our PAKE proof to simulate a proper reaction to a particular corruption-impersonation pattern.

Definition 5. Let $\text{KEM} = (\text{KGen}, \text{Encap}, \text{Decap})$ be a key encapsulation mechanism with key space \mathcal{K} . We define the IND-CPA game and the ANO-PCA game as in Fig. 2, relative to challenge bit b , and the respective advantage function of an adversary \mathcal{A} against KEM as

$$\begin{aligned} \text{Adv}_{\text{KEM}}^{\text{ANO-PCA}}(\mathcal{A}) &:= |\Pr[\text{ANO-PCA}^0(\mathcal{A})] - \Pr[\text{ANO-PCA}^1(\mathcal{A})]| \quad \text{and} \\ \text{Adv}_{\text{KEM}}^{\text{IND-CPA}}(\mathcal{A}) &:= |\Pr[\text{IND-CPA}^0(\mathcal{A})] - \Pr[\text{IND-CPA}^1(\mathcal{A})]| . \end{aligned}$$

In our proof we require that attackers could mistake any element of the public-key space for an honestly generated public key. Concretely, we formalize this below as a public-key uniformity game that asks the attacker to distinguish honestly generated public keys from uniformly random ones. This property was introduced as *fuzziness* in [BCP⁺23], where it was also proven for the post-quantum KEM Kyber. There also exist stronger (statistical) definitions – e.g., [BCJ⁺19] proved statistical public-key uniformity of discrete-log-based PKE schemes, due to public keys being uniformly chosen group elements.

Definition 6. Let $\text{KEM} = (\text{KGen}, \text{Encap}, \text{Decap})$ be a key encapsulation mechanism with public-key space \mathcal{PK} . We define the PKU game as in Fig. 3, relative to challenge bit b , and the respective advantage function of an adversary \mathcal{A} against KEM as

$$\text{Adv}_{\text{KEM}}^{\text{PKU}}(\mathcal{A}) := |\Pr[\text{PKU}^0(\mathcal{A})] - \Pr[\text{PKU}^1(\mathcal{A})]| .$$

Game IND-CPA^b 01 $(pk, sk) \leftarrow \$ \text{KGen}$ 02 $(c^*, K_0^*) \leftarrow \$ \text{Encap}(pk)$ 03 $K_1^* \xleftarrow{\text{unif}} \mathcal{K}$ 04 $b' \leftarrow \mathcal{A}(pk, c^*, K_b^*)$ 05 return $\llbracket b = b' \rrbracket$	
Game ANO-PCA^b 06 $(pk_0, sk_0) \leftarrow \$ \text{KGen}$ 07 $(pk_1, sk_1) \leftarrow \$ \text{KGen}$ 08 $(c^*, K^*) \leftarrow \$ \text{Encap}(pk_b)$ 09 $b' \leftarrow \mathcal{A}^{1\text{-PCO}_{c^*}}(pk_0, pk_1, c^*, K^*)$ 10 return $\llbracket b = b' \rrbracket$	1-PCO _{\mathcal{L}^*} (c, K) //only one query allowed 11 if $c \neq c^*$ 12 $K' \leftarrow \text{Decap}(sk_0, c)$ 13 return $\llbracket K = K' \rrbracket$ 14 else return \perp

Fig. 2: The {IND, ANO}-PCA security games for KEM, defined relative to challenge bit b : indistinguishability (IND-CPA) and anonymity (ANO-PCA) with plaintext-checking oracle 1-PCO. We make the convention that 1-PCO can only be queried once.

PKU^b(\mathcal{A}) 01 $(pk_0, sk_0) \leftarrow \$ \text{KGen}$ 02 $pk_1 \leftarrow \$ \mathcal{PK}$ 03 $b' \leftarrow \mathcal{A}(pk_b)$ 04 return $\llbracket b = b' \rrbracket$	PKU_n^b(\mathcal{A}) 05 for $j \in [n]$ 06 $(pk_{j,0}, sk_{j,0}) \leftarrow \$ \text{KGen}$ 07 $pk_{j,1} \xleftarrow{\text{unif}} \mathcal{PK}$ 08 $\mathbf{pk} \cdot \text{append}(pk_{j,b})$ 09 $b' \leftarrow \mathcal{A}(\mathbf{pk})$ 10 return $\llbracket b = b' \rrbracket$
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fig. 3: Public-key uniformity game PKU for KEM, and its multi-user counterpart PKU_n for n many users. Public-key uniformity is also known as fuzziness.

3 Multi-user notions for KEMs

Multi-user security notions were first introduced for public-key encryption in [BBM00] and then extended to IND-CPA security of KEMs in [GKP18]. To obtain a tight security proof for our PAKE protocol, we now define multi-user (and multi-challenge) counterparts for the previously introduced security notions. For IND-CPA and ANO-PCA, the respective notion (IND-CPA _{n, q_C} and ANO-PCA _{n, q_C}) models the setting where an adversary can ask for up to q_C many challenges for each of n many different key pairs. The adversary wins if it successfully attacks *any* of the up to nq_C challenges. We will also use a multi-user notion of public-key uniformity (PKU_n), where the adversary is tasked with distinguishing a vector of n many honestly generated public keys from a vector that consists of elements picked uniformly from the public key space.

We include generic reductions between single- and multi-user security. Looking ahead, the loss that occurs in these reductions reflects how session guessing would introduce reduction losses in our PAKE proof, were the multi-user notions replaced by single-user notions.

Definition 7 (Multi-user security notions for KEM). *Let KEM be a key encapsulation mechanism with public-key space \mathcal{PK} and key space \mathcal{K} . For integers n and q_C , we define the PKU_n game, the IND-CPA _{n, q_C} game and the ANO-PCA _{n, q_C} game as in Figures 3, 4 and 5, each relative to challenge*

Game $\text{IND-CPA}_{n,q_C}^b$	$\text{Chall}_{q_C}^b(j)$
01 for $i \in [n]$	06 $(c, K_0) \leftarrow \$ \text{Encap}(pk_j)$
02 $(pk_i, sk_i) \leftarrow \$ \text{KGen}$	07 $K_1 \xleftarrow{\text{unif}} \mathcal{K}$
03 $\mathbf{pk} \cdot \text{append}(pk_i)$	08 return (c, K_b)
04 $b' \leftarrow \mathcal{A}^{\text{Chall}}(\mathbf{pk})$	
05 return $\llbracket b = b' \rrbracket$	

Fig. 4: Multi-user indistinguishability game $\text{IND-CPA}_{n,q_C}$ for KEM, for n many users. Challenge oracle Chall can be queried at most q_C many times per user.

Game $\text{ANO-PCA}_{n,q_C}^b(\mathcal{A})$	$\text{Chall}_{q_C}^b(j)$	$1\text{-PCO}_{\mathcal{L}^*}(j, c, K)$	//once per j
01 for $j \in [n]$	08 $(c_0, K_0) \leftarrow \$ \text{Encap}(pk_{0,j})$	12 if $c \notin \mathcal{L}_j^*$	
02 $(pk_{0,j}, sk_{0,j}) \leftarrow \$ \text{KGen}$	09 $(c_1, K_1) \leftarrow \$ \text{Encap}(pk_{1,j})$	13 $K' \leftarrow \text{Decap}(sk_{0,j}, c)$	
03 $(pk_{1,j}, sk_{1,j}) \leftarrow \$ \text{KGen}$	10 $\mathcal{L}_j^* \leftarrow \mathcal{L}_j^* \cup \{c_b\}$	14 return $\llbracket K = K' \rrbracket$	
04 $\mathbf{pk}_0 \cdot \text{append}(pk_{0,j})$	11 return (c_b, K_b)	15 else return \perp	
05 $\mathbf{pk}_1 \cdot \text{append}(pk_{1,j})$			
06 $b' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathbf{pk}_0, \mathbf{pk}_1)$			
07 return $\llbracket b = b' \rrbracket$			

Fig. 5: Multi-user anonymity game $\text{ANO-PCA}_{n,q_C}$ for KEM, for n many users. The collection \mathcal{O} of \mathcal{A} 's oracles is $\mathcal{O} = \{1\text{-PCO}, \text{Chall}_{q_C}^b\}$. We make the same query restrictions and initialization conventions as in Fig. 4.

bit b , and the respective advantage function of an adversary \mathcal{A} against KEM as

$$\begin{aligned} \text{Adv}_{\text{KEM}}^{\text{IND-CPA}_{n,q_C}}(\mathcal{A}) &:= |\Pr[\text{IND-CPA}_{n,q_C}^0(\mathcal{A})] - \Pr[\text{IND-CPA}_{n,q_C}^1(\mathcal{A})]|, \\ \text{Adv}_{\text{KEM}}^{\text{ANO-PCA}_{n,q_C}}(\mathcal{A}) &:= |\Pr[\text{ANO-PCA}_{n,q_C}^0(\mathcal{A})] - \Pr[\text{ANO-PCA}_{n,q_C}^1(\mathcal{A})]| \text{ and} \\ \text{Adv}_{\text{KEM}}^{\text{PKU}^{(n)}}(\mathcal{A}) &:= |\Pr[\text{PKU}_n^0(\mathcal{A})] - \Pr[\text{PKU}_n^1(\mathcal{A})]|. \end{aligned}$$

It is known [GKP18, Lemma 3.2] that 'plain' IND-CPA security lifts generically to its multi-user counterpart with a loss of $n \cdot q_C$, where n is the number of users (the number of public keys) and q_C is the maximal number of challenge queries per public key. We now show that this also holds for anonymity for adversaries that also have access to a plaintext checking oracle (ANO-PCA) and public-key uniformity (PKU) (with a loss of n). The obtained generic bounds might be overly pessimistic for specific KEMs, considering that a KEM's underlying structure might allow for tighter reasoning.

Theorem 1. *Let KEM be a key encapsulation mechanism. For any $\text{ANO-PCA}_{n,q_C}$ adversary \mathcal{A} against KEM, there exists an ANO-PCA adversary \mathcal{B} against KEM such that*

$$\text{Adv}_{\text{KEM}}^{\text{ANO-PCA}_{n,q_C}}(\mathcal{A}) \leq n \cdot q_C \cdot \text{Adv}_{\text{KEM}}^{\text{ANO-PCA}}(\mathcal{B}).$$

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

Proof. We reduce single-user ANO-PCA anonymity of KEM to multi-user anonymity $\text{ANO-PCA}_{n,q_C}$, using a very similar hybrid argument. Let \mathcal{A} be an adversary in the $\text{ANO-PCA}_{n,q_C}$ game defined in Fig. 5.

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 \mathbf{pk}_1) to $b = 0$ (all challenges built using \mathbf{pk}_0). We will iterate over j/i , the number of public keys/queries per public key for which we will implement the change: In game $\mathbf{G}_{j,i}$, oracle $\text{Chall}(j')$ upon the i' -th query uses

- the respective public key $pk_{0,j'}$ from \mathbf{pk}_0 if $(j' < j)$ or if (both $j' = j$ and $i' < i$),
- the respective public key $pk_{1,j'}$ from \mathbf{pk}_1 if $j' > j$ or if (both $j' = j$ and $i' \geq i$).

Using the triangle inequality yields

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

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 ANO-PCA adversaries \mathcal{B}_{ji} to upper bound the summands $|\Pr[\mathbf{G}_{j,i}^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{j,i+1}^{\mathcal{A}} \Rightarrow 1]|$: \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 ANO-PCA 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 $\mathbf{pk}_0, \mathbf{pk}_1$ 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 $\mathbf{pk}_0, \mathbf{pk}_1$ and answers \mathcal{A} 's challenge queries as follows: upon the i' -th query to $\text{Chall}(j')$, \mathcal{B}_{ji} responds with

- a challenge constructed using the respective public key $pk_{0,j'}$ from \mathbf{pk}_0 if $j' < j$, or if both $j' = j$ and $i' < i$
- a challenge constructed using the respective public key $pk_{1,j'}$ from \mathbf{pk}_1 if $j' > j$, or if both $j' = j$ and $i' > i$
- its own challenge (c^*, K^*) if $j' = j$ and $i' = i$

For all vector positions except for j , \mathcal{B}_{ji} possesses the secret key belonging to pk_j , thereby being able to respond to all of \mathcal{A} 's respective 1-PCD queries. If \mathcal{A} queries the 1-PCD with index j , \mathcal{B} forwards the query to its own 1-PCD oracle. When \mathcal{A} outputs a guess to \mathcal{B}_{ji} , \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

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

The running time of \mathcal{B}_{ji} is about that of \mathcal{A} .

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

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

where \mathcal{B} stems from folding the adversaries \mathcal{B}_{ji} into a single one. The running time of \mathcal{B} is about that of \mathcal{A} .

□

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

$$\text{Adv}_{\text{KEM}}^{\text{PKU}_{n,qC}}(\mathcal{A}) \leq n \cdot \text{Adv}_{\text{KEM}}^{\text{PKU}}(\mathcal{B}).$$

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

Proof. We reduce PKU security of KEM to multi-user security anonymity PKU_n via a hybrid argument. Let \mathcal{A} be an adversary in the PKU_n experiment as defined in Fig. 3. 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 \mathbf{pk} are sampled using KGen, and the last $n - i$ many are sampled uniformly at random from \mathcal{PK} .

Using the triangle inequality yields

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

We now give single-user PKU adversaries \mathcal{B}_i to upper bound the summands $|\Pr[\mathbf{G}_i^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{G}_{i+1}^{\mathcal{A}} \Rightarrow 1]|$: \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 many public keys \mathbf{pk} by using random sampling from \mathcal{PK} for the first $i - 1$ many, inserting its own challenge public key pk^* at the i -th position, and using KGen for the positions $i + 1$ to n .

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

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

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

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

where \mathcal{B} stems from folding the adversaries \mathcal{B}_{j_i} into a single one. The running time of \mathcal{B} is about that of \mathcal{A} . □

4 The Protocol OCAKE

We describe a 3-message password-authenticated key exchange (PAKE) protocol based on an ideal cipher IC and an implicitly-rejecting key encapsulation mechanism KEM in Figure 6. The protocol achieves mutual authentication and AKE security according to the BPR model, which we prove in Section 6.

Two parties, the initiator (\mathcal{I}) and the responder (\mathcal{R}), share a common password pw and proceed in three phases. First, \mathcal{I} will generate a KEM key pair, encrypt the public key using the password,

Initiator \mathcal{I} (Client)	OCAKE Protocol	Responder \mathcal{R} (Server)
Password pw		Password pw
	Transmit Encrypted Public Key	
$k_{pw} \leftarrow \text{KDF}(pw)$ $(pk, sk) \leftarrow \$ \text{KGen}$ $apk \leftarrow \text{BC.enc}(k_{pw}, pk)$	\xrightarrow{apk}	$k_{pw} \leftarrow \text{KDF}(pw)$ $pk' \leftarrow \text{BC.dec}(k_{pw}, apk)$
	Establish Session Pre-Key	
$K' \leftarrow \text{Decap}(sk, c)$	\xleftarrow{c}	$(c, K) \leftarrow \$ \text{Encap}(pk')$
$tag_2 \leftarrow \text{H}(pw, apk, pk, c, K', "i")$	$\xleftarrow{tag_1}$	$tag_1 \leftarrow \text{H}(pw, apk, pk', c, K, "r")$
	$\xrightarrow{tag_2}$	
	Key Confirmation, Key Derivation	
$tag'_1 \leftarrow \text{H}(pw, apk, pk, c, K', "r")$ if $tag'_1 = tag_1$ $SK \leftarrow \text{KDF}'(tag_1, K')$ output SK and accept terminate		$tag'_2 \leftarrow \text{H}(pw, apk, pk', c, K, "i")$ if $tag'_2 = tag_2$ $SK' \leftarrow \text{KDF}'(tag_1, K)$ output SK' and accept terminate

Fig. 6: The OCAKE protocol, using a key encapsulation mechanism $\text{KEM} = (\text{KGen}, \text{Encap}, \text{Decap})$ and a block cipher $\text{BC} = (\text{BC.enc}, \text{BC.dec})$ that is modeled as an ideal cipher in the proof. Messages c and tag_2 are part of the same round, so can be transmitted together, making this a three-round protocol. The second tag is used to extend the protocol to achieve mutual authentication and is not needed for our security definition.

and send the encrypted public key apk to \mathcal{R} . After receipt, \mathcal{R} uses the password to recover the public key, then computes an encapsulation c and pre-key K . As response, \mathcal{R} sends the encapsulation c and a responder tag tag_1 to \mathcal{I} .

On receipt, \mathcal{I} decapsulates the ciphertext to obtain a pre-key K' and compares the received tag to the one it derives from its own state. If the tags match, \mathcal{I} outputs a session key SK derived from the pre-key. For key confirmation, \mathcal{I} also computes an initiator tag and sends it to \mathcal{R} .

The received tag is checked by \mathcal{R} against its own state and if it matches, \mathcal{R} outputs a session key SK' derived from its pre-key. Under the correctness of KEM, both parties only output a session key if and only if both parties used the same password, in which case the two session keys SK and SK' are identical.

5 Security Model

Our security analysis is based on the BPR model for authenticated key exchange [BPR00]: security of a protocol Π is modeled through a security experiment in which the attacker interacts with oracles that represent honest parties (**Execute** and **Send**) as well as oracles that represent leakage of secret material (**Reveal** and **Corrupt**), and wins if it can distinguish an established session key from random. The involved oracles are described in more detail in Fig. 7.

Query	Return Value	Description
Execute (P, i, P', j)	(apk, c, tag_1, tag_2)	Passive attack: Return transcript of an honest protocol execution between parties P and P' , using the i th/ j th session of P/P' .
Send ($P, i, msg, flow$)	msg'	Active attack: Send message msg to the oracle representing honest party 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 "un-fresh".
Corrupt (P, PWD')	$PWD[P, :]$	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 ^{b} (P, i)	$SK[P, i]/SK^{\$}$	Session key challenge: Attack i th session of party P . Only for fresh, accepting instances. Returns either real or random session key depending on challenge bit b .
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 $SK \in SK$.
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. 7: 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 to when attacking the OCAKE protocol.

To exclude trivial attacks from consideration, [BPR00] define a freshness condition (Definition 9 below) that permits revealing a key on one side, and then testing the other (partnered) side, where partnered is defined as follows:

Definition 8 (Partnering). *Two instances $(P, i), (P', j)$ are partnered iff both instances have accepted (i.e. reached an **accept** 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 trivially derivable regardless of the protocol’s nature. 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 9 (Freshness with Forward Secrecy). *Suppose that the adversary made exactly one **Test** query, and it was to party P and instance i . We say session i of party P is **unfresh** if at any time, there was a **Reveal** query to instance (P, i) above or the instance (P', j) that it is partnered with. We also say the session is **unfresh** if both the following conditions hold:*

- Before the **Test** query, there was a **Corrupt** query on the test session’s holder P or its partnered peer P' .
- 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 session of (P, i) is only considered **fresh** if neither of these conditions are met.*

```

Experiment  $\text{Exp}_\Pi^{\text{BPR}}(\mathcal{A})$ 
16  $b \xleftarrow{\text{unif}} \{0, 1\}$ 
17  $b' \leftarrow \mathcal{A}^{\mathcal{O}^b}(\mathcal{P})$ 
18 return  $\llbracket b = b' \rrbracket$ 

```

Fig. 8: The BPR security game for active adversaries. $\mathcal{O}^b =$ indicates the collection of oracles $\{\text{Execute}, \text{KDF}, \text{H}, \text{KDF}', \text{IC. enc}, \text{IC. dec}, \text{Send}, \text{Reveal}, \text{Corrupt}, \text{Test}^b\}$. Here, \mathcal{P} is the party set.

Definition 10 (Key indistinguishability of PAKE). *Let Π be a PAKE protocol. We say that an adversary \mathcal{A} , run in experiment $\text{Exp}_\Pi^{\text{BPR}}$, wins if it correctly guesses the bit according to which the test query was defined and if the **Test** query was issued for a party (P, i) that has terminated and is fresh (see Definition 9). We define the advantage of \mathcal{A} against a PAKE protocol Π as*

$$\text{Adv}_\Pi^{\text{BPR}}(\mathcal{A}) := |\Pr[\text{Exp}_\Pi^{\text{BPR}}(\mathcal{A}) \Rightarrow 1] - 1/2| .$$

Our modification of OCAKE uses key confirmation tags in both directions. While only the responder tag actually is needed for our security proof, we additionally include an initiator tag – following the ‘add client-to-server authentication’ (AddCSA) paradigm [BPR00] – to achieve explicit mutual authentication.

Definition 11 (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.*

6 Security of OCAKE

Our main result is Theorem 3 below which relates forward security of OCAKE to security of the used KEM, in the combined Random Oracle (RO) and Ideal Cipher (IC) model. During its proof, we consider an adversary playing the BPR security game for our protocol OCAKE.

Theorem 3 (Tight security of OCAKE in the combined RO and IC model 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 OCAKE[KEM, KDF, KDF', H, BC], issuing at most n_a many *Send* queries (i.e. active attacks), n_p many *Execute* queries (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_{RO} many queries to its respective random oracles. Let $n_s := n_a + n_p$ be the total number of sessions. Then there exist a multi-user-IND-CPA adversary \mathcal{B}^{IND} , a multi-user-ANO-PCA adversary \mathcal{B}^{ANO} and a multi-user-PKU adversary \mathcal{B}^{PKU} against KEM such that*

$$\begin{aligned} \text{Adv}_{\text{OCAKE}}^{\text{BPR}}(\mathcal{A}) &\leq \frac{n_a}{|\mathcal{D}|} + \text{Adv}_{\text{KEM}}^{\text{PKU}(q_{IC.dec} + n_s)}(\mathcal{B}^{\text{PKU}}) + 2 \cdot \text{Adv}_{\text{KEM}}^{\text{ANO-PCA}(q_{IC.dec} + n_s, n_a + 1)}(\mathcal{B}^{\text{ANO}}) \\ &\quad + 2 \cdot \text{Adv}_{\text{KEM}}^{\text{IND-CPA}(n_s, n_a + 1)}(\mathcal{B}^{\text{IND}}) + \frac{3 \cdot q_{IC}^2}{2 \cdot |\mathcal{PK}|} + 2 \cdot n_s \cdot \delta \\ &\quad + q_{RO} \cdot n_s \cdot \left(\frac{2}{|\mathcal{SK}|} + \frac{2}{|\mathcal{K}|} \right) + q_{RO}^2 \cdot \left(\frac{1}{2 \cdot |\mathcal{T}|} + \frac{1}{2 \cdot |\mathcal{K}_{pw}|} \right) \end{aligned}$$

and the running time of \mathcal{B}^{IND} , \mathcal{B}^{ANO} , and \mathcal{B}^{PKU} is about that of \mathcal{A} .

Theorem 4 (Generic Security of OCAKE 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 OCAKE[KEM, KDF, KDF', H, BC], issuing at most n_a 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 the random oracles. Then there exist single-user-IND-CPA adversary \mathcal{B}^{IND} a single-user-ANO-PCA adversary \mathcal{B}^{ANO} and a single-user-PKU adversary \mathcal{B}^{PKU} against KEM such that*

$$\begin{aligned} \text{Adv}_{\text{OCAKE}}^{\text{BPR}}(\mathcal{A}) &\leq \frac{n_a}{|\mathcal{D}|} + (q_{IC.dec} + n_s) \cdot \text{Adv}_{\text{KEM}}^{\text{PKU}}(\mathcal{B}^{\text{PKU}}) \\ &\quad + 2 \cdot n_s \cdot (n_a + 1) \cdot \text{Adv}_{\text{KEM}}^{\text{IND-CPA}}(\mathcal{B}^{\text{IND}}) \\ &\quad + 2 \cdot (q_{IC.dec} + n_s) \cdot (n_a + 1) \cdot \text{Adv}_{\text{KEM}}^{\text{ANO-PCA}}(\mathcal{B}^{\text{ANO}}) \\ &\quad + q_{RO} \cdot n_s \cdot \left(\frac{2}{|\mathcal{SK}|} + \frac{2}{|\mathcal{K}|} \right) + q_{RO}^2 \cdot \left(\frac{1}{2 \cdot |\mathcal{T}|} + \frac{1}{2 \cdot |\mathcal{K}_{pw}|} \right) \\ &\quad + n_s \cdot \delta + \frac{3 \cdot q_{IC}^2}{2 \cdot |\mathcal{PK}|} \end{aligned}$$

and the running time of \mathcal{B}^{IND} , \mathcal{B}^{ANO} , and \mathcal{B}^{PKU} is about that of \mathcal{A} .

Theorem 4 follows directly from plugging the generic multi-user reductions in 3 into Theorem 3, we therefore proceed by proving the ‘tight’ Theorem 3. Intuitively, the proof of Theorem 3 reflects three security goals. We show that (SG1) the adversary can test at most one password per session with which it actively interferes, (SG2) honest protocol runs do not leak a significant amount 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. Since we achieve forward secrecy, goal (SG3) is also achieved for sessions where the adversary knows the password, as long as the session is not actively attacked.⁴ Pseudo-code for the BPR oracles is shown in Figure 9. Amongst the other oracles, Fig. 9

sketches the Send^i oracles, where i indicates the flow number 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. During that argument, we need to exclude the bad-case that protocol messages could stem from multiple passwords due to collisions. Game hops \mathbf{G}_1 to \mathbf{G}_5 aim at eliminating this bad-case. After that, we address security goals SG2 and SG3 by eliminating leakage on the password and the session key with game hops \mathbf{G}_9 to \mathbf{G}_{11} . Game hops \mathbf{G}_6 to \mathbf{G}_8 are preparation for these changes.

6.1 Original Security Game

Original game \mathbf{G}_0 . The first game is the original BPR security game, with oracles Send and Execute answering queries according to the protocol (see Fig. 9).

$$\text{Adv}_{\text{OCAKE}}^{\text{BPR}}(\mathcal{A}) = |\Pr[\mathbf{G}_0(\mathcal{A}) \Rightarrow 1]| - 1/2 .$$

In the following game hops, we will use the notational convention that $\text{Adv}_i := |\Pr[\mathbf{G}_i(\mathcal{A}) \Rightarrow 1]|$.

6.2 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 Collision in Key Generation In this game, we abort whenever there are at least two sessions where the same ephemeral key pair (pk, sk) is sampled by the KEM key generation. Let η_{KGen} be the collision probability of KGen. Since games \mathbf{G}_0 and \mathbf{G}_1 are identical unless a collision occurs, we have that:

$$|\text{Adv}_0 - \text{Adv}_1| = \Pr[\text{KDFCo11}] \leq n_s^2 \cdot \eta_{\text{KGen}}$$

Game \mathbf{G}_2 : 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.

⁴ We use these intuitive security goals to structure the proof, however the only formal security goal is indistinguishability of session keys.

<p>Initialization</p> <p>01 for $(P, P') \in \mathcal{P} \times \mathcal{P}$</p> <p>02 $pw \leftarrow \\$PW()$</p> <p>03 $PWD[\{P, P'\}] \xleftarrow{\text{set}} pw$</p> <p>Execute$(P, i, P', j)$</p> <p>04 $apk \leftarrow \text{Send}^0(P, i, \perp)$</p> <p>05 $c, tag_1 \leftarrow \text{Send}^1(P', j, apk)$</p> <p>06 $tag_2 \leftarrow \text{Send}(P, i, (c, tag_1))$</p> <p>07 $\text{Send}(P', j, tag_2)$</p> <p>08 $\text{MANIP}[\{P, : \}] \leftarrow \text{false}$</p> <p>09 return (apk, c, tag_2, tag_1)</p> <p>Corrupt(P, PWD')</p> <p>10 $PWD_P \leftarrow PWD[\{P, : \}]$</p> <p>11 $\text{CRPT}[\{P, : \}] \leftarrow \text{true}$</p> <p>12 $PWD[\{P, : \}] \leftarrow PWD'[\{P, : \}]$</p> <p>13 return PWD_P</p> <p>Reveal(P, i)</p> <p>14 $\text{RVL}[(P, i)] \leftarrow \text{true}$</p> <p>15 return $\text{SK}[(P, i)]$</p> <p>Test^{b}(P, i)</p> <p>16 $SK_0 \leftarrow K[(P, i)]$</p> <p>17 $SK_1 \xleftarrow{\text{unif}} \mathcal{SK}$</p> <p>18 if $(\text{CRPT}[\{P, P'\}]$ and $\text{MANIP}[(P, i)])$</p> <p>19 or if $(\text{RVL}[(P, i)]$ or $\text{RVL}[(P', j)])$</p> <p>20 or if $(\text{SK}[(P, i)] = \perp)$: return \perp</p> <p>21 else: return SK_b</p>	<p>Send⁰(P, i, msg)</p> <p>22 $\text{MANIP}[(P, i)] \leftarrow \text{true}$</p> <p>23 $k_{pw} \leftarrow \text{KDF}(pw)$</p> <p>24 $(pk, sk) \leftarrow \\$KGen$</p> <p>25 $apk \leftarrow \text{IC. enc}(k_{pw}, pk)$</p> <p>26 return apk</p> <p>Send¹(P, i, msg)</p> <p>27 $\text{MANIP}[(P, i)] \leftarrow \text{true}$</p> <p>28 $apk \xleftarrow{\text{parse}} msg$</p> <p>29 $k_{pw} \leftarrow \text{KDF}(pw)$</p> <p>30 $pk' \leftarrow \text{IC. dec}(k_{pw}, apk)$</p> <p>31 $(c, K) \leftarrow \\$Encap(pk')$</p> <p>32 $tag_1 \leftarrow \text{H}(pw, apk, pk', c, K, "r")$</p> <p>33 return (c, tag_1)</p> <p>Send²(P, i, msg)</p> <p>34 $\text{MANIP}[(P, i)] \leftarrow \text{true}$</p> <p>35 $c, tag_1 \xleftarrow{\text{parse}} msg$</p> <p>36 $K' \leftarrow \text{Decap}(sk, c)$</p> <p>37 if $tag_1 = \text{H}(pw, apk, pk, c, K', "r")$:</p> <p>38 $tag_2 \leftarrow \text{H}(pw, apk, pk, c, K', "i")$</p> <p>39 $SK \leftarrow \text{KDF}'(tag_1, K')$</p> <p>40 $K[(P, i)] \xleftarrow{\text{set}} SK$</p> <p>41 return tag_2</p> <p>42 else: return \perp</p> <p>Send³(P, i, msg)</p> <p>43 $\text{MANIP}[(P, i)] \leftarrow \text{true}$</p> <p>44 $tag_2 \xleftarrow{\text{parse}} msg$</p> <p>45 if $tag_2 = \text{H}(pw, apk, pk', c, K, "i")$:</p> <p>46 $SK \leftarrow \text{KDF}'(tag_1, K)$</p> <p>47 $K[(P, i)] \xleftarrow{\text{set}} SK$</p>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fig. 9: The oracles in the security game for OCAKE. PWD is the dictionary of the parties' passwords and CRPT, MANIP and RVL indicate the corruption status of a session. Password generation in the initialization phase (**Initialization**) is modeled using the long-lived key generator PW .

	Change	Reasoning	Loss
\mathbf{G}_1	Prevent KGen collisions	KGen entropy	$n_s^2 \cdot \eta_{\text{KGen}}$
\mathbf{G}_2	Prevent KDF collisions	Search Bound	$\frac{q_{\text{KDF}}^2}{2 \cdot \mathcal{K}_{pw} }$
\mathbf{G}_3	IC lazy sampling w/ abort	Search Bound	$\frac{q_{\text{IC}}^2}{2 \cdot \mathcal{PK} }$
\mathbf{G}_4	Prevent IC collisions	Search Bound	$\frac{q_{\text{IC}}^2}{ \mathcal{PK} }$
\mathbf{G}_5	Prevent resp. tag collision	Search Bound	$\frac{q_{\text{H}}^2}{2 \cdot \mathcal{T} }$
\mathbf{G}_6	Sample IC using KGen	pk uniformity	$(q_{\text{IC.dec}} + n_s)\text{-PKU}$
\mathbf{G}_7	Abort on corr pw	Password Guessing	$\frac{n_a}{ \mathcal{D} } + \Delta \Pr[\text{corrPW}]$
\mathbf{G}_8	Honest c : Replace Decap by responder's pre-key	Correctness	$n_s \cdot \delta$
\mathbf{G}_9	Randomize public-key pk	Anonymity	$(q_{\text{IC.dec}} + n_s, n_a + 1)\text{-ANO-PCA}$
\mathbf{G}_{10}	Randomize pre-key K	Indistinguishability	$(n_s, n_a + 1)\text{-IND-CPA}$
\mathbf{G}_{11}	Randomize tags tag_1, tag_2	Random Oracle	$\frac{q_{\text{H}} \cdot n_s}{ \mathcal{K} }$
\mathbf{G}_{12}	Randomize session key SK	Random Oracle	$\frac{q_{\text{KDF}'} \cdot n_s}{ \mathcal{SK} }$

Fig. 10: Overview of all game changes and their associated loss. $\Delta \Pr[\text{corrPW}]$ is equal to the sum of all following game hops.

We now keep a list of all previous queries to the KDF oracle by recording all input-output pairs (pw, k_{pw}) . Let KDFCo11 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:

$$\text{KDFCo11} : k_{pw} = k_{pw'} \text{ for queries } k_{pw} \leftarrow \text{KDF}(pw), k_{pw'} \leftarrow \text{KDF}(pw').$$

In game \mathbf{G}_1 , we abort whenever this event occurs. Let q_{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: $\Pr[\text{KDFCo11}] \leq \frac{q_{\text{KDF}}^2}{2 \cdot |\mathcal{K}_{pw}|}$. Since games \mathbf{G}_1 and \mathbf{G}_2 are identical unless KDFCo11 occurs, the distance of the adversary's success probability is bounded:

$$|\mathbf{Adv}_1 - \mathbf{Adv}_2| = \Pr[\text{KDFCo11}] \leq \frac{q_{\text{KDF}}^2}{2 \cdot |\mathcal{K}_{pw}|}$$

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}_3 : Simulate Ideal Cipher with abort 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. For every record, we also record the direction of the query that first created the record, with a label "enc" for encryption and "dec" for decryption. We give a pseudocode description in Figure Fig. 11. Sampling this way is done in preparation for games \mathbf{G}_6 and \mathbf{G}_9 , where we replace ideal cipher outputs with public keys generated using KGen.

$\text{IC. enc}(k_{pw}, pk)$ 01 if \exists record (k_{pw}, pk, apk, \star) : 02 return apk 03 else 04 $apk' \xleftarrow{\text{unif}} \mathcal{PK}$ 05 if \exists record $(k_{pw}, \star, apk', \star)$: abort 06 create record $(k_{pw}, pk, apk', \text{"enc"})$ 07 return apk'	$\text{IC. dec}(k_{pw}, apk)$ 08 if \exists record (k_{pw}, pk, apk) : 09 return pk 10 else 11 $pk' \xleftarrow{\text{unif}} \mathcal{PK}$ 12 if \exists record $(k_{pw}, pk', \star, \star)$: abort 13 create record $(k_{pw}, pk', apk, \text{"dec"})$ 14 return pk'
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fig. 11: The simulated ideal cipher sampling with abort. The star (\star) matches any value in that field.

Game \mathbf{G}_3 is identical to \mathbf{G}_2 unless it aborts in line 5 of Figure Fig. 11. The probability of this occurring can be bounded using a standard collision bound in the total number of ideal cipher queries q_{IC} and the size of the public-key space $|\mathcal{PK}|$ and therefore:

$$|\mathbf{Adv}_3 - \mathbf{Adv}_2| \leq \frac{q_{\text{IC}}^2}{2 \cdot |\mathcal{PK}|}.$$

Game \mathbf{G}_4 : 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 $k_{pw} \neq k'_{pw}$

$$\begin{aligned} \text{ICCo111} : (pk, apk) = (pk', apk') \text{ for two queries:} \\ (\text{either } apk \leftarrow \text{IC. enc}(k_{pw}, pk) \text{ or } pk \leftarrow \text{IC. dec}(k_{pw}, apk)) \\ \text{and } (\text{either } (apk' \leftarrow \text{IC. enc}(k'_{pw}, pk') \text{ or } pk' \leftarrow \text{IC. dec}(k'_{pw}, apk')) \end{aligned}$$

To give an example, an adversary sending an apk with this property 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.⁵ Formally, this would imply that there were two queries to the ideal cipher such that for $k_{pw} \neq k'_{pw}$:

$$\text{ICCo112} : apk = apk' \text{ for queries: } apk \leftarrow \text{IC. enc}(k_{pw}, pk), apk' \leftarrow \text{IC. enc}(k'_{pw}, pk')$$

In game \mathbf{G}_4 , we abort whenever ICCo111 or ICCo112 occur. We define the event ICCo11 where $\Pr[\text{ICCo11}] := \Pr[\text{ICCo111} \vee \text{ICCo112}]$. 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)

⁵ 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.

$\text{IC. enc}_{\mathbf{G}_4}(k_{pw}, pk)$	$\text{IC. enc}_{\mathbf{G}_4}(k_{pw}, pk)$	$\text{IC. dec}_{\mathbf{G}_4}(k_{pw}, apk)$	$\text{IC. dec}_{\mathbf{G}_4}(k_{pw}, apk)$
01 if \exists record (k_{pw}, pk, apk, \star) :		10 if \exists record (k_{pw}, pk, apk, \star) :	
02 return apk		11 return pk	
03 else		12 else	
04 $apk' \xleftarrow{\text{unif}} \mathcal{PK}$		13 $pk' \xleftarrow{\text{unif}} \mathcal{PK}$	
05 if \exists record $(k_{pw}, \star, apk', \star)$: abort		14 if \exists record $(k_{pw}, pk', \star, \star)$: abort	
06 if \exists record (\star, pk, apk', \star) : ICColl1		15 if \exists record (\star, pk', apk, \star) : ICColl1	
07 if \exists record $(\star, \star, apk', \text{"enc"})$: ICColl2		16 create record $(k_{pw}, pk', apk, \text{"dec"})$	
08 create record $(k_{pw}, pk, apk', \text{"enc"})$		17 return pk'	
09 return apk'			

Fig. 12: The simulated ideal cipher. In game \mathbf{G}_4 , the game aborts whenever there is a collision in the ideal cipher that would allow the adversary to test two passwords.

q_{IC} and the size of the ideal cipher domain $|\mathcal{PK}|$, since it requires sampling. In game \mathbf{G}_4 , we abort whenever **ICColl** occurs. Since games \mathbf{G}_4 and \mathbf{G}_4 are identical unless **ICColl** occurs, it holds that

$$|\mathbf{Adv}_4 - \mathbf{Adv}_4| = \Pr[\text{ICColl}] \leq \frac{q_{\text{IC}}^2}{2 \cdot |\mathcal{PK}|} + \frac{q_{\text{IC. enc}}^2}{2 \cdot |\mathcal{PK}|} \leq \frac{q_{\text{IC}}^2}{|\mathcal{PK}|}.$$

Game \mathbf{G}_5 : Abort on Responder Tag Collision We now create a record for all queries to \mathbf{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}_5 aborts. The probability of **ROColl** occurring is bounded using a standard collision bound given by the number of queries $q_{\mathbf{H}}$ to \mathbf{H} and the size of the tag space T : $\Pr[\text{ROColl}] \leq \frac{q_{\mathbf{H}}^2}{2 \cdot |T|}$. Since games \mathbf{G}_5 and \mathbf{G}_5 are identical unless **ROColl** occurs, it holds that:

$$|\mathbf{Adv}_5 - \mathbf{Adv}_5| = \Pr[\text{ROColl}] \leq \frac{q_{\mathbf{H}}^2}{2 \cdot |T|}.$$

For every responder tag output by the \mathbf{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}_6 : Sample Ideal Cipher Outputs Using KEM Key Generation. In game \mathbf{G}_6 , we replace the way the simulated 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}_9 . The change is depicted in Fig. 13.

$\text{IC.dec}_{\mathbf{G}_6}(k_{pw}, apk)$	$\text{IC.dec}_{\mathbf{G}_6}(k_{pw}, apk)$
01 if \exists record (k_{pw}, pk, apk)	
02 return pk	
03 else	
04 $pk' \xleftarrow{\text{unif}} \mathcal{PK}$	$pk' \xleftarrow{\$} \text{KGen}$
05 if \exists record (k_{pw}, pk', \star) : abort	
06 if \exists record (\star, pk', apk) : abort	
07 create record (k_{pw}, pk', apk)	
08 return pk'	

Fig. 13: The simulated ideal cipher now samples using the KEM’s key generation algorithm KGen instead of uniformly at random from the domain \mathcal{PK} .

We will now argue that an adversary that can distinguish game \mathbf{G}_6 from \mathbf{G}_6 can be used to attack the n -public-key-uniformity (PKU_n) property of the underlying KEM for $n = q_{\text{IC.dec}} + n_s$, by means of a reduction \mathcal{B}^{PKU} . Let \mathcal{A} be the adversary running either in game \mathbf{G}_6 or \mathbf{G}_6 , issuing at most $q_{\text{IC.dec}}$ many queries to the ideal cipher decryption oracle. We define adversary \mathcal{B}^{PKU} against the PKU_n experiment (defined in Fig. 3) as follows (for the sake of formality, we give the pseudo-code of \mathcal{B}^{PKU} in Fig. 14):

<u>Adversary $\mathcal{B}^{\text{PKU}}(\mathbf{pk})$</u>	<u>$\text{IC.dec}(k_{pw}, apk)$</u>
01 $\text{pkIndex} = 0$	05 if \exists record (k_{pw}, pk, apk) :
02 $b \xleftarrow{\text{unif}} \{0, 1\}$	06 return pk
03 $b' \leftarrow \mathcal{A}^{\mathcal{O}^b}(\mathbf{pk})$	07 else
04 output $b'_{\text{PKU}} := [b = b']$	08 $pk^* \leftarrow \mathbf{pk}[\text{pkIndex}]$ //pk^* is real or random
	09 $\text{pkIndex} += 1$
	10 if \exists record (k_{pw}, pk^*, \star) : abort
	11 if \exists record (\star, pk^*, apk) : abort
	12 create record (k_{pw}, pk^*, apk)
	13 return pk'

Fig. 14: PKU adversary \mathcal{B}^{PKU} , used to reason about the hop from game \mathbf{G}_6 to \mathbf{G}_6 . Adversary \mathcal{A} has access to oracles $\mathcal{O} = \{\text{KDF}, \text{KDF}', \text{IC.enc}, \text{IC.dec}, \text{Execute}, \text{Send}, \text{Reveal}, \text{Corrupt}\}$.

\mathcal{B}^{PKU} receives a vector of challenge public keys \mathbf{pk} of dimension n from its PKU_n challenger, where $n := q_{\text{IC.dec}} + n_s$. (Depending on the challenger’s bit b_{PKU} , \mathbf{pk} is generated using KGen or drawn uniformly at random from \mathcal{PK} .) \mathcal{B}^{PKU} samples an own challenge bit b' , runs \mathcal{A} and answers \mathcal{A} ’s queries to the Oracles \mathcal{H} , IC.enc , KDF, KDF' , Reveal , Send , Execute , and $\text{Test}^{b'}$ according to the oracles in \mathbf{G}_6 . When simulating ideal cipher decryption queries, \mathcal{B}^{PKU} embeds the challenge public keys from its input vector \mathbf{pk} : Upon a query to oracle IC.dec , instead of sampling an output like game \mathbf{G}_6 (see line 4 of Fig. 11), \mathcal{B}^{PKU} uses the next value in \mathbf{pk} . In case a query is repeated, it repeats the respective public key. \mathcal{B}^{PKU} needs to produce at most

$q_{\text{IC.dec}} + n_s$ many outputs for IC.dec , one for each direct query ($q_{\text{IC.dec}}$), and one per protocol session (n_s). When \mathcal{A} outputs a guess b , \mathcal{B}^{PKU} checks if $b = b'$ and in that case returns $b'_{\text{PKU}} := 1$ as its own output bit, otherwise, \mathcal{B}^{PKU} returns $b'_{\text{PKU}} := 0$.

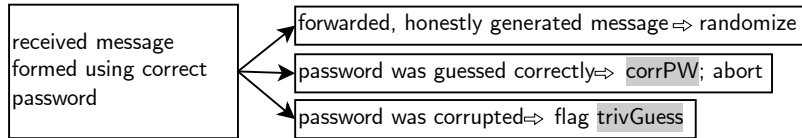
\mathcal{B}^{PKU} perfectly simulates \mathbf{G}_6 when run with KGen-generated public keys, and \mathbf{G}_6 when run with uniform public keys. Since \mathcal{B}^{PKU} uses at most $n = q_{\text{IC.dec}} + n_s$ many public keys in total, the difference between \mathcal{A} 's winning probabilities in games \mathbf{G}_6 and \mathbf{G}_6 is upper bounded by the n -uniformity advantage of \mathcal{B}^{PKU} against KEM:

$$|\text{Adv}_6 - \text{Adv}_6| \leq \text{Adv}_{\text{KEM}}^{\text{PKU}(q_{\text{IC.dec}} + n_s)}$$

6.3 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 6.4. For these randomizations to go unnoticed, we need to rule out the case that the adversary sent messages constructed using the correct password. Therefore, whenever the adversary makes a Send query, we start to check if the correct password was used. If such a query occurs, there are three possible reasons:

1. **trivGuess** The adversary obtained the password by *corrupting* one of the parties involved in the session. In that case, we raise the flag **trivGuess** for that session and continue the protocol without applying the changes of the following games, i.e., without randomization. In cases where **trivGuess** is raised during a session, in between subsequent Send queries to the same session, it is relevant for which message this flag is first raised. We denote the event that **trivGuess** is raised for the flow- i message as **trivGuess_i**.
2. **forward** The adversary *forwards* a message that was generated honestly by a previous query to 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 6.4. The game detects the event by keeping a record of all honestly generated transcripts.
3. **corrPW** 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. 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.



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.⁶

⁶ There is a small subtlety here: in an edge-case discussed later, we make the change also when **trivGuess** is raised in the middle of a session.

Game \mathbf{G}_7 : Abort on Correct Password. We consider two cases of correct password guesses:

Authenticated Public Key (apk) Consider the event where the adversary sends an apk built from the correct password *and* the respective message was indeed generated by the adversary, i.e., the message was not honestly generated by a previous call to **Send**. We'll call this event **apkCorrPw**. The game detects this event using a record of honestly generated messages and the ideal cipher encryption records. The changes in the previous games guarantee that the record is unique.

Responder Tag (tag_1) Consider the event where the adversary sends a valid tag_1 built from the correct password *and* the respective message was indeed generated by the adversary. We'll call this event **tagCorrPw**. The game detects this 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 input used to create it which contains a password. By the changes made in previous games, there is at most one record for this tag, uniquely determining the *password* that was used / tested in this query.

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.

In case the adversary submits a tag_2 (i.e. flow 3 message) formed using the correct password, one of two things has happened: either the tag matches the responder's transcript, and we are in the forwarding case, or it does not, in which case the responder rejects. Therefore, we do not have to consider this event a correct guess.

We let game \mathbf{G}_7 abort whenever **corrPW** occurs. Since both games proceed identically unless **corrPW** occurs, we have

$$|\mathbf{Adv}_7 - \mathbf{Adv}_7| = \Pr[\mathbf{corrPW}_{\mathbf{G}_7}]$$

We track how the probability of event **corrPW** changes throughout the sequence of games, and finish by bounding its probability in game \mathbf{G}_{12} .

6.4 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, 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 does not matter anymore 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 reduction.

Game \mathbf{G}_8 : Do Not Decapsulate Honest Ciphertexts. In game \mathbf{G}_8 , whenever there is a flow 2 query where the message was honestly generated by a matching session, we do not decapsulate to obtain the pre-key. Instead, if the message was generated by a matching session, we use the pre-key that was generated by that instance. Adversarially generated messages as well as ones that are forwarded from a non-matching session are decapsulated as before. This step is done in preparation for the reductions in the following two game hops.

$\text{Send}_{\mathbf{G}_8}^2(P, i, \text{msg})$	$\text{Send}_{\mathbf{G}_8}^2(P, i, \text{msg})$
01 $c, \text{tag}_1 \xleftarrow{\text{parse}} \text{msg}$	
02 $K' \leftarrow \text{Decap}(sk, c)$	if forward: $K' \leftarrow$ responder's key K
03	else: $K' \leftarrow \text{Decap}(sk, c)$
04 if $\text{tag}_1 = \text{H}(pw, apk, pk, c, K', "r")$:	
05 $\text{tag}_2 \leftarrow \text{H}(pw, apk, pk, c, K', "i")$	
06 $SK \leftarrow \text{KDF}'(\text{tag}_1, K')$	
07 $K[(P,i)] \xleftarrow{\text{set}} SK$	
08 return tag_2	
09 else: return \perp	

Fig. 15: In game \mathbf{G}_8 , whenever **forward** occurs (i.e., c is a matching responder's honest ciphertext) we use the responder's pre-key K instead of decapsulating c .

Games \mathbf{G}_8 and \mathbf{G}_8 are indistinguishable unless a correctness error occurred in game \mathbf{G}_8 and therefore:

$$|\Pr[\text{corrPW}_{\mathbf{G}_8}] - \Pr[\text{corrPW}_{\mathbf{G}_8}]| = |\mathbf{Adv}_8 - \mathbf{Adv}_8| = n_s \cdot \delta$$

Game \mathbf{G}_9 : Randomize Encapsulation Public Key. In the first randomization step, we make the following change for all queries to the **Send** 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 16). We will now argue that an adversary noticing this change can be used to attack the multi-user anonymity property $\text{ANO-PCA}_{n, q_C}$ where $n := q_{\text{IC.dec}} + n_s$ and $q_C := n_a + 1$. 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** 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.

$\text{Send}_{\mathbf{G}_9}^1(P, i, \text{msg})$	$\text{Send}_{\mathbf{G}_9}^1(P, i, \text{msg})$
01 $apk \xleftarrow{\text{parse}} \text{msg}$	
02 $k_{pw} \leftarrow \text{KDF}(pw)$	
03 $pk' \leftarrow \text{IC.dec}(k_{pw}, apk)$	
04	if $\text{PK}[(k_{pw}, apk)] \neq \perp$: $pk'_s \leftarrow \text{PK}[(k_{pw}, apk)]$
	else: $(pk'_s, sk'_s) \leftarrow \text{\$KGen}$
	$\text{PK}[(k_{pw}, apk)] \xleftarrow{\text{set}} pk'_s$
05 $(c, K) \leftarrow \text{\$Encap}(pk')$	$(c, K) \leftarrow \text{\$Encap}(pk'_s)$
06 $\text{tag}_1 \leftarrow \text{H}(pw, apk, pk', c, K, "r")$	
07 return c, tag_1	

Fig. 16: Game \mathbf{G}_9 : Randomizing public key in Send^1 queries. The dictionary **PK** is a book-keeping tool introduced in game \mathbf{G}_9 to ensure consistency of replays.

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. We define

adversary $\mathcal{B}_0^{\text{ANO}}$ against the ANO-PCA $_{n,q_C}$ experiment (defined in Fig. 5) as follows (for the sake of formality, we give the pseudo-code of $\mathcal{B}_0^{\text{ANO}}$ in Fig. 17):

$\mathcal{B}_0^{\text{ANO}}$ receives two vectors of challenge public keys $(\mathbf{pk}_0, \mathbf{pk}_1)$ of dimension $n = q_{\text{IC.dec}} + n_s$, and can query its challenge oracle **Chall**, provided by its ANO-PCA $_{n,q_C}$ challenger, at most $q_C = n_a + 1$ many times. (Depending on the challenger's bit, the challenges are generated using either \mathbf{pk}_0 or \mathbf{pk}_1 .) $\mathcal{B}_0^{\text{ANO}}$ samples an own challenge bit b' , runs \mathcal{A} and answers \mathcal{A} 's queries to the Oracles **H**, **IC.enc**, **KDF**, **KDF'**, **Send**³, **Reveal**, and **Test** ^{b'} according to the oracles in \mathbf{G}_9 . **Execute** queries are answered using **Send** as before.

On ideal cipher decryption queries, $\mathcal{B}_0^{\text{ANO}}$ embeds the challenge public keys contained in \mathbf{pk}_0 (see Fig. 17). When \mathcal{A} queries **Send**⁰, $\mathcal{B}_0^{\text{ANO}}$ uses one of the challenge public keys in \mathbf{pk}_0 .

Whenever \mathcal{A} queries the **Send**¹ oracle and **trivGuess** has not been raised, 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}_4 , there must exist $j \in [n]$ s.t. $pk' = pk_{0,j}$. Then, to answer the query, $\mathcal{B}_0^{\text{ANO}}$ queries **Chall**(j) to receive a challenge (c^*, K^*) , outputs c^* to \mathcal{A} and uses K^* as K (see Fig. 17). Ciphertexts returned by the **Send**¹ oracle are then either encapsulations under the public key $pk_{0,j}$ or under $pk_{1,j}$, depending on the challenge bit in the ANO-PCA $_{n,q_C}$ game. Note that since \mathcal{A} can replay an *apk* value in each of the n_a many sessions, the same public key will sometimes be used to obtain multiple challenges and $q_C = n_a + 1$.

Whenever \mathcal{A} queries **Send**², there is an edge-case to consider: In case the adversary causes the **trivGuess** flag to be raised *before* **Send**² is queried but *after* **Send**⁰ is, the adversary is able to forge a tag for an arbitrary ciphertext under the challenge public key chosen in **Send**⁰. To learn the pre-key needed to complete the simulation of the initiator, $\mathcal{B}_0^{\text{ANO}}$ queries the 1-PCO oracle using the pre-key K' matching the record of tag_1 . If that query returns true, the instance accepts and with $SK \leftarrow \text{KDF}'(tag_1, K')$, and rejects if not.

When \mathcal{A} outputs a guess b , $\mathcal{B}_0^{\text{ANO}}$ 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.

$\mathcal{B}_0^{\text{ANO}}$ perfectly simulates \mathbf{G}_9 when run in the ANO-PCA $_{n,q_C}$ -game with challenge bit 0, \mathbf{G}_9 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}_9 and \mathbf{G}_9 is upper bounded by the respective ANO-PCA $_{n,q_C}$ advantage of $\mathcal{B}_0^{\text{ANO}}$ against KEM:

$$|\text{Adv}_9 - \text{Adv}_9| \leq \text{Adv}_{\text{KEM}}^{\text{ANO-PCA}_{(q_{\text{IC.dec}} + n_s, n_a + 1)}}(\mathcal{B}_0^{\text{ANO}})$$

To keep track of the change in the probability of $\text{Pr}[\text{corrPW}]$, we can slightly adapt the reduction $\mathcal{B}_0^{\text{ANO}}$: our new reduction $\mathcal{B}_1^{\text{ANO}}$ behaves exactly like $\mathcal{B}_0^{\text{ANO}}$ except for its output: $\mathcal{B}_1^{\text{ANO}}$ returns 1 if **corrPW** occurred, and otherwise 0.

$$|\text{Pr}[\text{corrPW}_{\mathbf{G}_9}] - \text{Pr}[\text{corrPW}_{\mathbf{G}_9}]| \leq \text{Adv}_{\text{KEM}}^{\text{ANO-PCA}_{(q_{\text{IC.dec}} + n_s, n_a + 1)}}(\mathcal{B}_1^{\text{ANO}})$$

Game \mathbf{G}_{10} : Randomize Session Pre-Key. For all queries to the **Send** or **Execute** oracles where flag **trivGuess** is not raised before the query, 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 18. This change makes the pre-key independent of the ciphertext and the password for all fresh sessions. We now argue that an adversary noticing this change can be used to attack the indistinguishability property of the KEM. We define adversary $\mathcal{B}_0^{\text{IND}}$ against the IND-CPA $_{n,q_C}$ experiment (defined in Fig. 4) as follows (for the sake of formality, we give the pseudo-code of $\mathcal{B}_0^{\text{IND}}$ in Fig. 19):

<p><u>Adversary $\mathcal{B}_0^{\text{ANO}}(\mathbf{pk}_0, \mathbf{pk}_1)$</u></p> <p>01 $\text{pkIndex} = 0$</p> <p>02 $b \xleftarrow{\text{unif}} \{0, 1\}$</p> <p>03 $b' \leftarrow \mathcal{A}^{\mathcal{O}^b}()$</p> <p>04 $b'_{\text{ANO}} := [b = b']$</p> <p>05 output b'_{ANO}</p> <p><u>Send⁰(P, i, msg)</u></p> <p>14 if trivGuess_0 : return $\text{Send}^0(P, i, \text{msg})_{\mathbf{G}_7}$</p> <p>15 $k_{pw} \leftarrow \text{KDF}(pw)$</p> <p>16 $pk \leftarrow \mathbf{pk}_0[\text{pkIndex}]$ //$pk' \leftarrow \\$\text{KGen}$</p> <p>17 $\text{pkIndex} += 1$</p> <p>18 $apk \leftarrow \text{IC.enc}_{k_{pw}}(pk)$ //get challenge pk</p> <p>19 return apk</p> <p><u>Send¹(P, i, msg)</u></p> <p>20 if trivGuess_0 or trivGuess_1 : return $\text{Send}^1(P, i, \text{msg})_{\mathbf{G}_7}$</p> <p>21 $apk \xleftarrow{\text{parse}} \text{msg}$</p> <p>22 $k_{pw} \leftarrow \text{KDF}(pw)$</p> <p>23 $pk' \leftarrow \text{IC.dec}(k_{pw}, apk)$</p> <p>24 find j s.t. $pk' = \mathbf{pk}_0[j]$ //IC returned challenge pk from \mathbf{pk}_0</p> <p>25 $(c, K) \leftarrow \text{Chall}(j)$ //(c, K) $\leftarrow \text{Encap}(pk')$</p> <p>26 $\text{tag}_1 \leftarrow \text{H}(pw, apk, pk', c, K, "r")$</p> <p>27 return c, tag_1</p> <p><u>Send²(P, i, msg)</u></p> <p>28 if trivGuess_0 or trivGuess_1 : return $\text{Send}^2(P, i, \text{msg})_{\mathbf{G}_7}$</p> <p>29 $c, \text{tag}_1 \xleftarrow{\text{parse}} \text{msg}$</p> <p>30 if forward : $K' \leftarrow$ responder's key K</p> <p>31 else if \exists record $\text{tag}_1 = \text{H}(pw, apk, pk, c, K_A, "r")$: //event trivGuess_2</p> <p>32 find j s.t. $pk = \mathbf{pk}_0[j]$ //see flow 0 to see this exists</p> <p>33 if $[1\text{-PCO}(j, c, K_A) \Rightarrow \text{true}]$: $K' \leftarrow K_A$</p> <p>34 else: return \perp</p> <p>35 else: return \perp</p> <p>36 if $\text{tag}_1 = \text{H}(pw, apk, pk, c, K', "r")$:</p> <p>37 $\text{tag}_2 \leftarrow \text{H}(pw, apk, pk, c, K', "i")$</p> <p>38 $SK \leftarrow \text{KDF}'(\text{tag}_1, K)$</p> <p>39 $\text{K}[(P, i)] \xleftarrow{\text{set}} SK$</p> <p>40 return tag_2</p> <p>41 else: return \perp</p>	<p><u>IC.dec(k_{pw}, apk)</u></p> <p>06 if \exists record (k_{pw}, pk, apk) return pk :</p> <p>07 else</p> <p>08 $pk' \leftarrow \mathbf{pk}_0[\text{pkIndex}]$ //$pk' \leftarrow \\$\text{KGen}$</p> <p>09 $\text{pkIndex} += 1$</p> <p>10 if \exists record (k_{pw}, pk', \star) : abort</p> <p>11 if \exists record (\star, pk', apk) : abort</p> <p>12 create record (k_{pw}, pk', apk)</p> <p>13 return pk'</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fig. 17: ANO-PCA $_{n, qc}$ adversary $\mathcal{B}_0^{\text{ANO}}$, used to reason about the hop from game \mathbf{G}_9 to \mathbf{G}_7 . The collection \mathcal{O} of \mathcal{A} 's oracles is $\mathcal{O} = \{\text{KDF}, \text{KDF}', \text{IC.enc}, \text{IC.dec}, \text{Execute}, \text{Send}, \text{Reveal}, \text{Corrupt}\}$. In case of corruption prior to each query, $\mathcal{B}_0^{\text{ANO}}$ follows the protocol according to the oracles in game \mathbf{G}_7 , with the exception of the edge case shown in lines 31 to 35.

$\text{Send}_{\mathbf{G}_{10}}^1(P, i, \text{msg})$	$\text{Send}_{\mathbf{G}_{10}}^1(P, i, \text{msg})$
01 $apk \xleftarrow{\text{parse}} \text{msg}$	
02 $k_{pw} \leftarrow \text{KDF}(pw)$	
03 $pk' \leftarrow \text{IC.dec}(k_{pw}, apk)$	
04 if $\text{PK}[(k_{pw}, apk)] \neq \perp$:	
05 $pk'_s \leftarrow \text{PK}[(k_{pw}, apk)]$	
06 else:	
07 $(pk'_s, sk_s) \xleftarrow{\$} \text{KGen}$	
08 $\text{PK}[(k_{pw}, apk)] \xleftarrow{\text{set}} pk'_s$	
09 $(c, K) \xleftarrow{\$} \text{Encap}(pk'_s)$	
10	$K_s \xleftarrow{\text{unif}} \mathcal{K}$
11 $tag_1 \leftarrow \text{H}(pw, apk, pk', c, K, "r")$	$tag_1 \leftarrow \text{H}(pw, apk, pk', c, K_s, "r")$
12 return c, tag_1	

Fig. 18: In game \mathbf{G}_{10} , the pre-key set after querying **Send** or **Execute** is sampled independently of the password and the previous messages. Due to the change in game 8, this also randomizes the initiator side and we also write $K'_s \leftarrow K_s$.

<u>Adversary $\mathcal{B}_0^{\text{IND}}$</u>	<u>$\text{Send}^1(P, i, \text{msg})$</u>
01 input pk	07 $k_{pw} \leftarrow \text{KDF}(pw)$
02 $pkIndex = 0$	08 if \exists record $\text{PK}[(k_{pw}, apk)]$: //handle replays
03 $b \xleftarrow{\text{unif}} \{0, 1\}$	09 $pk'_s \leftarrow \text{PK}[(k_{pw}, apk)]$
04 $b' \leftarrow \mathcal{A}^b(pk)$	10 else:
05 $b'_{\text{IND}} := [b = b']$	11 $pk'_s \leftarrow pk[pkIndex]$
06 output b'_{IND}	12 $pkIndex += 1$
	13 $\text{PK}[(k_{pw}, apk)] \xleftarrow{\text{set}} pk'_s$
	14 find j s.t. $pk'_s = pk_j$
	15 $(c, K) \leftarrow \text{Chall}(j)$ // K is real-or-random w.r.t. c
	16 $tag_1 \leftarrow \text{H}(pw, apk, pk', c, K, "r")$
	17 return c, tag_1

Fig. 19: IND-CPA $_{n,q_C}$ adversary $\mathcal{B}_0^{\text{IND}}$, used to reason about the hop from game \mathbf{G}_{10} to \mathbf{G}_{10} . The set of oracles is $\mathcal{O} = \{\text{KDF}, \text{KDF}', \text{IC.enc}, \text{IC.dec}, \text{Execute}, \text{Send}, \text{Reveal}, \text{Corrupt}\}$.

$\mathcal{B}_0^{\text{IND}}$ receives a vector of challenge public keys pk , of dimension $n = n_s$ and can query its challenge oracle **Chall**, provided by its IND-CPA $_{n,q_C}$ challenger, at most $q_C = n_a + 1$ many times. $\mathcal{B}_0^{\text{IND}}$ samples a challenge bit b' , runs \mathcal{A} and answers \mathcal{A} 's queries to the Oracles **H**, **IC.enc**, **KDF**, **Reveal**, **Send⁰**, **Send²**, and **Test^{b'}** according to the oracles in \mathbf{G}_{10} .

On **Send¹** queries, $\mathcal{B}_0^{\text{IND}}$ issues a **Chall**(j) query to its own challenger receive (c^*, K^*) , where j is the index of the public key which it uses to answer the query. If **trivGuess** has been raised, $\mathcal{B}_0^{\text{IND}}$ generates a key pair and continues the protocol honestly without inserting any challenges in this session. 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. When \mathcal{A} outputs a guess b ,

$\mathcal{B}_0^{\text{IND}}$ checks if $b = b'$. In the case that $b = b'$, it returns 1 as its own output bit, otherwise, it returns 0.

$\mathcal{B}_0^{\text{IND}}$ perfectly simulates \mathbf{G}_{10} when run in the $\text{IND-CPA}_{n_s, n_a+1}$ - game with challenge bit 0, \mathbf{G}_{10} when run with challenge bit 1, and returns 1 if the adversary wins. Therefore, the difference between \mathcal{A} 's winning probabilities in games \mathbf{G}_{10} and \mathbf{G}_{10} is upper bounded by the respective $\text{IND-CPA}_{n_s, n_a+1}$ advantage of $\mathcal{B}_0^{\text{IND}}$ against KEM:

$$|\text{Adv}_{10} - \text{Adv}_{10}| \leq \text{Adv}_{\text{KEM}}^{\text{IND-CPA}_{(n_s, n_a+1)}}(\mathcal{B}_0^{\text{IND}}) + n_s \cdot \delta$$

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

$$|\Pr[\text{corrPW}_{\mathbf{G}_{10}}] - \Pr[\text{corrPW}_{\mathbf{G}_{10}}]| \leq \text{Adv}_{\text{KEM}}^{\text{IND-CPA}_{(n_s, n_a+1)}}(\mathcal{B}_1^{\text{IND}})$$

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}_{11} : Randomize Tags. 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 20.

$\text{Send}_{\mathbf{G}_{11}}^1(P, i, msg)$	$\text{Send}_{\mathbf{G}_{11}}^1(P, i, msg)$	$\text{Send}_{\mathbf{G}_{11}}^2(P, i, msg)$	$\text{Send}_{\mathbf{G}_{11}}^2(P, i, msg)$
01 $apk \xleftarrow{\text{parse}} msg$		14 $c, tag_1 \xleftarrow{\text{parse}} msg$	
02 $k_{pw} \leftarrow \text{KDF}(pw)$		15 if forward :	
03 $pk' \leftarrow \text{IC.dec}(k_{pw}, apk)$		16 $K'_s \leftarrow \text{responder's key } K_s$	
04 if $\text{PK}[(k_{pw}, apk)] \neq \perp$:		17 $tag'_{1s} \leftarrow \text{responder's tag } tag_{1s}$	
05 $pk'_s \leftarrow \text{PK}[(k_{pw}, apk)]$		18 else:	
06 else:		19 $K'_s \leftarrow \text{Decap}(sk', c)$	
07 $(pk'_s, sk_s) \leftarrow \text{KGen}$		20 $tag_{1s} \leftarrow \text{H}(pw, apk, pk, c, K'_s, "r")$	
08 $\text{PK}[(k_{pw}, apk)] \xleftarrow{\text{set}} pk'_s$		21 if $tag_1 = \text{H}(pw, apk, pk, c, K'_s, "r")$:	
09 $(c, K) \leftarrow \text{Encap}(pk'_s)$		22 if $tag_1 = tag'_{1s}$:	
10 $K_s \xleftarrow{\text{unif}} \mathcal{K}$		23 $tag_2 \leftarrow \text{H}(pw, apk, pk, c, K'_s, "i")$	
11 $tag_1 \leftarrow \text{H}(pw, apk, pk', c, K_s, "r")$		24 $tag_{2s} \xleftarrow{\text{unif}} \mathcal{T}$	
12 $tag_{1s} \xleftarrow{\text{unif}} \mathcal{T}$		25 $SK \leftarrow \text{KDF}'(tag_1, K'_s)$	
13 return c, tag_1	return c, tag_{1s}	26 $SK \leftarrow \text{KDF}'(tag_{1s}, K'_s)_s$	
		27 $\text{K}[(P, i)] \xleftarrow{\text{set}} SK$	
		28 return tag_2	return tag_{2s}

Fig. 20: Randomizing tags. The domain of the random oracle H is \mathcal{T} . The tag check for the initiator tag is modified in an equivalent fashion.

Let TagQueried be the event that the adversary has queried the random oracle H on input $(pw, apk, pk', c, K_s, "r")$ or $(pw, apk, pk, c, K'_s, "i")$. We argue that due to H being a random oracle, games \mathbf{G}_{11} and \mathbf{G}_{11} are indistinguishable to the adversary unless TagQueried occurs. Therefore, if \mathcal{A} can issue at most q_{H} queries to the random oracle H , we have

$$\Pr[\text{corrPW}_{\mathbf{G}_{11}}] - \Pr[\text{corrPW}_{\mathbf{G}_{11}}] = |\text{Adv}_{11} - \text{Adv}_{11}| \leq \Pr[\text{TagQueried}] \leq \frac{q_{\text{H}} \cdot n_s}{|\mathcal{K}|}$$

6.5 Randomizing Session Key (SG3)

Game \mathbf{G}_{12} : Randomize Session Key. Finally, we replace the final session key for all **Send** and **Execute** queries where flag `trivGuess` did not occur 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.

$\text{Send}_{\mathbf{G}_{12}}^2(P, i, msg)$	$\text{Send}_{\mathbf{G}_{12}}^2(P, i, msg)$	$\text{Send}_{\mathbf{G}_{12}}^3(P, i, msg)$	$\text{Send}_{\mathbf{G}_{12}}^3(P, i, msg)$
01 $c, tag_1 \xleftarrow{\text{parse}} msg$		14 $tag_2 \xleftarrow{\text{parse}} msg$	
02 if forward :		15 if forward :	
03 $K'_s \leftarrow$ responder's key K_s		16 $tag'_{2s} \leftarrow$ responder's tag tag_{2s}	
04 $tag'_{1s} \leftarrow$ responder's tag tag_{1s}		17 else:	
05 else:		18 $tag_{2s} \leftarrow \text{H}(pw, apk, pk, c, K'_s, "i")$	
06 $K'_s \leftarrow \text{Decap}(sk', c)$		19 if $tag_2 = tag'_{2s}$	
07 $tag_{1s} \leftarrow \text{H}(pw, apk, pk, c, K'_s, "r")$		20 $SK \leftarrow \text{KDF}'(tag_{1s}, K_s)$	
08 ...		21 $K[(P,i)] \xleftarrow{\text{set}} SK$	$K[(P,i)] \xleftarrow{\text{set}} SK_s$
09 if $tag_1 = tag'_{1s}$:			
10 $tag_{2s} \xleftarrow{\text{unif}} \mathcal{T}$			
11 $SK \leftarrow \text{KDF}'(tag_1, K'_s)$	$SK_s \xleftarrow{\text{unif}} \mathcal{SK}$		
12 $K[(P,i)] \xleftarrow{\text{set}} SK$	$K[(P,i)] \xleftarrow{\text{set}} SK_s$		
13 return tag_{2s}			

Fig. 21: In game \mathbf{G}_{12} , the final session key is randomized. To remain consistent, the initiator uses the responder's session key.

Let SKQueried be the event that the adversary has queried $\text{KDF}(tag_2, K_s)$. In game \mathbf{G}_{12} , we abort whenever this occurs. We argue that due to KDF' being a random oracle, games \mathbf{G}_{12} and \mathbf{G}_{12} are indistinguishable to the adversary unless SKQueried occurs. Therefore, for an adversary that can issue at most q'_{KDF} queries to the random oracle KDF' and n_s potential session keys, we have that

$$\Pr[\text{corrPW}_{\mathbf{G}_{12}}] - \Pr[\text{corrPW}_{\mathbf{G}_{12}}] = |\mathbf{Adv}_{12} - \mathbf{Adv}_{12}| \leq \Pr[\text{SKQueried}] \leq \frac{q'_{\text{KDF}} \cdot n_s}{|\mathcal{SK}|}$$

After this change, the adversary's **Test** query will always respond with a uniformly random value that is independent of the challenge bit. The winning probability of \mathcal{A} in game \mathbf{G}_{12} is therefore reduced to that of random guessing:

$$\mathbf{Adv}_{12} = \frac{1}{2}.$$

Bounding Correct Password Event. All protocol messages are now independent of the respective password for all fresh sessions, meaning they do not give the adversary any information about those passwords. However, the adversary can still attempt a password guess by picking a password from the password space, using it in a **Send** query, and observing if the game aborts. We can bound the probability of a correct 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 n_a many send queries, we get the bound: $\Pr[\text{corrPW}_{\mathbf{G}_{12}}] \leq \frac{n_a}{|\mathcal{D}|}$. Collecting the probabilities,

we can now bound the probability of event `corrPW` occurring in game 7:

$$\begin{aligned} \Pr[\text{corrPW}_{\mathbf{G}_7}] &\leq \sum_{i=7}^{11} |\Pr[\text{corrPW}_{\mathbf{G}_i}] - \Pr[\text{corrPW}_{\mathbf{G}_{i+1}}]| + \Pr[\text{corrPW}_{\mathbf{G}_{12}}] \\ &\leq \frac{n_a}{|\mathcal{D}|} + \text{Adv}_{\text{KEM}}^{\text{ANO-PCA}_{(q_{\text{IC.dec}}+n_s, n_a+1)}}(\mathcal{B}_1^{\text{ANO}}) + \text{Adv}_{\text{KEM}}^{\text{IND-CPA}_{(n_s, n_a+1)}}(\mathcal{B}_1^{\text{IND}}) \\ &\quad + \frac{q_{\text{H}} \cdot n_s}{|\mathcal{K}|} + \frac{q_{\text{KDF}'} \cdot n_s}{|\mathcal{SK}|} \end{aligned}$$

6.6 Total bound on BPR security

To wrap up the proof, we can now bound the BPR advantage of an adversary against the OCAKE protocol, using the triangle inequality. We will also fold the two anonymity adversaries $\mathcal{B}_0^{\text{ANO}}$ and $\mathcal{B}_1^{\text{ANO}}$ into one (\mathcal{B}^{ANO}), as well as the two indistinguishability adversaries $\mathcal{B}_0^{\text{IND}}$ and $\mathcal{B}_1^{\text{IND}}$ (\mathcal{B}^{IND}). We also consolidate the random oracles into RO .

$$\begin{aligned} \text{Adv}_{\text{OCAKE}}^{\text{BPR}}(\mathcal{A}) &\leq |\text{Adv}_0 - \text{Adv}_{11}| - \frac{1}{2} \\ &\leq \sum_{i=1}^{11} |\text{Adv}_i - \text{Adv}_{i+1}| - \frac{1}{2} \\ &= n_s^2 \cdot \eta_{\text{KGen}} + \frac{q_{\text{KDF}}^2}{2 \cdot |\mathcal{K}_{pw}|} + \frac{q_{\text{IC}}^2}{2 \cdot |\mathcal{PK}|} + \frac{q_{\text{IC}}^2}{|\mathcal{PK}|} + \frac{q_{\text{H}}^2}{2 \cdot |\mathcal{T}|} \\ &\quad + \underbrace{\Pr[\text{apkCorrPW}] + \Pr[\text{tagCorrPW}]}_{=\Pr[\text{corrPW}]} + \\ &\quad + \text{Adv}_{\text{KEM}}^{\text{PKU}_{(q_{\text{IC.dec}}+n_s)}}(\mathcal{B}^{\text{PKU}}) \\ &\quad + \text{Adv}_{\text{KEM}}^{\text{ANO-PCA}_{(q_{\text{IC.dec}}+n_s, n_a+1)}}(\mathcal{B}_0^{\text{ANO}}) \\ &\quad + \text{Adv}_{\text{KEM}}^{\text{IND-CPA}_{(n_s, n_a+1)}}(\mathcal{B}_0^{\text{IND}}) \\ &\quad + \frac{q_{\text{H}} \cdot n_s}{|\mathcal{K}|} + \frac{q_{\text{KDF}'} \cdot n_s}{|\mathcal{SK}|} + n_s \cdot \delta + \frac{1}{2} - \frac{1}{2} \\ &= \frac{n_a}{|\mathcal{D}|} + n_s^2 \cdot \eta_{\text{KGen}} + \frac{q_{\text{KDF}}^2}{2 \cdot |\mathcal{K}_{pw}|} + \frac{q_{\text{IC}}^2}{2 \cdot |\mathcal{PK}|} + \frac{q_{\text{IC}}^2}{|\mathcal{PK}|} \\ &\quad + \frac{q_{\text{H}}^2}{2 \cdot |\mathcal{T}|} + 2 \cdot \frac{q_{\text{H}} \cdot n_s}{|\mathcal{K}|} + \text{Adv}_{\text{KEM}}^{\text{PKU}_{(q_{\text{IC.dec}}+n_s)}}(\mathcal{B}^{\text{PKU}}) \\ &\quad + \text{Adv}_{\text{KEM}}^{\text{ANO-PCA}_{(q_{\text{IC.dec}}+n_s, n_a+1)}}(\mathcal{B}_0^{\text{ANO}}) + \text{Adv}_{\text{KEM}}^{\text{ANO-PCA}_{(q_{\text{IC.dec}}+n_s, n_a+1)}}(\mathcal{B}_1^{\text{ANO}}) \\ &\quad + \text{Adv}_{\text{KEM}}^{\text{IND-CPA}_{(n_s, n_a+1)}}(\mathcal{B}_0^{\text{IND}}) + \text{Adv}_{\text{KEM}}^{\text{IND-CPA}_{(n_s, n_a+1)}}(\mathcal{B}_1^{\text{IND}}) \\ &\quad + n_s \cdot \delta + 2 \cdot \frac{q_{\text{KDF}'} \cdot n_s}{|\mathcal{SK}|} \\ &= \frac{n_a}{|\mathcal{D}|} + \text{Adv}_{\text{KEM}}^{\text{PKU}_{(q_{\text{IC.dec}}+n_s)}}(\mathcal{B}^{\text{PKU}}) + 2 \cdot \text{Adv}_{\text{KEM}}^{\text{ANO-PCA}_{(q_{\text{IC.dec}}+n_s, n_a+1)}}(\mathcal{B}^{\text{ANO}}) \\ &\quad + 2 \cdot \text{Adv}_{\text{KEM}}^{\text{IND-CPA}_{(n_s, n_a+1)}}(\mathcal{B}^{\text{IND}}) + \frac{3 \cdot q_{\text{IC}}^2}{2 \cdot |\mathcal{PK}|} + n_s \cdot \delta \\ &\quad + q_{\text{RO}} \cdot n_s \cdot \left(\frac{2}{|\mathcal{SK}|} + \frac{2}{|\mathcal{K}|} \right) + q_{\text{RO}}^2 \cdot \left(\frac{1}{2 \cdot |\mathcal{T}|} + \frac{1}{2 \cdot |\mathcal{K}_{pw}|} \right) + n_s^2 \cdot \eta_{\text{KGen}} \end{aligned}$$

References

- ABR⁺21. Michel Abdalla, Manuel Barbosa, Peter B. Rønne, Peter Y. A. Ryan, and Petra ala. 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-pace-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.
- Ber22. Daniel J. Bernstein. Multi-ciphertext security degradation for lattices. *Cryptology ePrint Archive*, Report 2022/1580, 2022. <https://eprint.iacr.org/2022/1580>.
- 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 secure against 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.
- BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.
- 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.
- DKL⁺18. Léo Ducas, Eike Kiltz, Tancrede 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.
- HV22. Loïs Huguenin-Dumittan and Serge Vaudenay. On IND-qCCA security in the ROM and its applications - CPA security is sufficient for TLS 1.3. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 613–642, Trondheim, Norway, May 30 – June 3, 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 697711, 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.
- MX23. Varun Maram and Keita Xagawa. Post-quantum anonymity of Kyber. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 13940 of *Lecture Notes in Computer Science*, pages 3–35, Atlanta, GA, USA, May 7–10, 2023. Springer, Heidelberg, Germany.
- 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>.
- PZ23. Jiaxin Pan and Runzhi Zeng. A generic construction of tightly secure password-based authenticated key exchange. *Cryptology ePrint Archive*, Paper 2023/1334, 2023. <https://eprint.iacr.org/2023/1334>.
- 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.
- Xag22. Keita Xagawa. Anonymity of NIST PQC round 3 KEMs. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 551–581, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany.
- Zha19. Mark Zhandry. How to record quantum queries, and applications to quantum indistinguishability. 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.