Password-Authenticated Public-Key Encryption

Tatiana Bradley[1], Jan Camenisch[2], Stanislaw Jarecki[1], Anja Lehmann[3], Gregory Neven[2], and Jiayu Xu[1]

[1] University of California, Irvine. Email: {tebradle@,sjarecki@,jiayux@}uci.edu.
[2] Dfinity. Email: {gregory@,jan@}dfinity.org.
[3] IBM Research – Zurich. Email: anj@zurich.ibm.com.

Abstract. We introduce password-authenticated public-key encryption (PAPKE), a new cryptographic primitive. PAPKE enables secure end-to-end encryption between two entities without relying on a trusted third party or other out-of-band mechanisms for authentication. Instead, resistance to man-in-the-middle attacks is ensured in a human-friendly way by authenticating the public key with a shared password, while preventing offline dictionary attacks given the authenticated public key and/or the ciphertexts produced using this key.

Our contributions are three-fold. First, we provide property-based and universally composable (UC) definitions for PAPKE, with the resulting primitive combining CCA security of public-key encryption (PKE) with password authentication. Second, we show that PAPKE implies Password-Authenticated Key Exchange (PAKE), but the reverse implication does not hold, indicating that PAPKE is a strictly stronger primitive than PAKE. Indeed, PAPKE implies a two-flow PAKE which remains secure if either party re-uses its state in multiple sessions, e.g. due to communication errors, thus strengthening existing notions of PAKE security. Third, we show two highly practical UC PAPKE schemes: a generic construction built from CCA-secure and anonymous PKE and an ideal cipher, and a direct construction based on the Decisional Diffie-Hellman assumption in the random oracle model.

Finally, applying our PAPKE-to-PAKE compiler to the above PAPKE schemes we exhibit the first 2-round UC PAKE's with efficiency comparable to (unauthenticated) Diffie-Hellman Key Exchange.

1 Introduction

A well-known Achilles' heel of end-to-end encryption is the distribution and trustworthiness of long-term cryptographic keys [38]. In particular, it is extremely hard for end users to judge the authenticity of public keys. They can therefore easily be tricked into encrypting the data under a wrong key and thereby lose all security. If the exchange of keys is facilitated by a third party such as a certificate authority or a service provider, as is the case for most public-key infrastructures (PKIs) as well as for end-to-end encrypted messengers such as Signal, WhatsApp, or iMessage, users need to trust that third party to provide the correct keys. Indeed, if a service provider is able to substitute its own keys

for those of the intended recipients, it can mount a man-in-the-middle (MITM) attack and decrypt all subsequent communication.

An article in The Guardian [23] describes this trust required in the service provider and its capability of striking MITM attacks as a "backdoor" and a "security loophole" in the encryption scheme used by WhatsApp. This characterization was repudiated in an open letter signed by over seventy cryptographers and security experts [37], stating that this is not a "backdoor", but simply how cryptography works. While technically correct, this explanation is not very satisfactory from an end-user's perspective and prompts the question: *should cryptography work like that?* Is there really no way to protect encrypted communication between end users from such MITM and key substitution attacks?

Ad-hoc solutions against MITM attacks. Many approaches to preventing man-in-the-middle attacks in the context of secure end-to-end communication exist, but they either rely on trusted third parties, or on mostly ad-hoc solutions built on top of conventional encryption schemes, aiming to allow end-users to verify the correctness of public keys. None of these approaches provides the degree of usability and security that one can hope for, and which our solution provides.

Trust-on-first-use, as commonly used by Secure Shell (SSH), reduces the likelihood of MITM attacks, but cannot completely prevent them. The *web of trust* concept [35] as used, for example, in Pretty Good Privacy (PGP), establishes a distributed trust model via individual vetting: it requires users to endorse associations of public keys to specific people, and to endorse other people as trusted endorsers. Even though this approach was popular in the early days of cryptography, it was never widely adopted, possibly because of the strong level of involvement it requires from users to inspect each others' keys and to issue endorsements.

Today, the most common method to establish trust in end users' public keys is to let users manually verify a hash value of keys, known as a key *fingerprint*, using an out-of-band channel. Fingerprints are often represented in human-friendly formats to ease verification, e.g., as digits [39], pronounceable strings [26], ASCII art [34], or QR codes [39], but they require either physical proximity of communication partners (QR codes) or they are tedious to verify.[4] A crucial problem with key fingerprints is the far-from-optimal trade-off between security and usability: Strong fingerprints with 60 decimal or 32 hexadecimal digits are simply too long to verify by hand. Shorter fingerprints are more human-friendly but are vulnerable to preimage attacks, allowing an adversary to generate a key with the same fingerprint. A recent study comparing different manual key verification mechanisms found that all were subject to attacks whose success rates ranged between 6% and 72% [36].

Introducing a New Tool. We propose a new cryptographic primitive, *Password-Authenticated Public-Key Encryption (PAPKE)*, which authenticates an encryption public key using human-memorable passwords, but does so in a way which

[4] Users also struggle with the notion of key fingerprints, e.g. *all* Telegram users in one study [5] believed the fingerprint to be either the encryption key or a ciphertext.

is not subject to offline dictionary attacks given the authenticated public key or the ciphertexts encrypted using that key.

More precisely, PAPKE modifies the notion of public-key encryption so that both the *key generation* and the *encryption* algorithms take as an additional input *a password*, i.e. an arbitrary human-memorable string. The semantics of this password input is that Alice, when generating her public key, implicitly authenticates and "locks" this key with a password, and to encrypt a message, Bob must use a matching password to "unlock" the authenticated public key correctly. The correctness guarantee is that Alice decrypts the message encrypted by Bob *if* Bob encrypted it using the same password that Alice used in key generation. The notion of password-authentication of the public key which PAPKE enforces is the following: If a man-in-the-middle attacker substitutes Alice's public key with its own, the confidentiality of messages which Bob encrypts under this key is guaranteed as long as the adversary fails to guess the password shared by Alice and Bob. Crucially, the attacker must guess this password at the time it creates the substituted public key, and the eventual leakage of the password after generation of the adversarial key has no impact on encryption security.

PAPKE thus enables end-to-end secure communication without relying on a trusted party or exchanging long fingerprints on an out-of-band channel, and instead it bootstraps security from a short human-memorizable password.

PAPKE and Offline Dictionary Attacks. The challenge of password-based schemes is obtaining strong security based on weak secrets. In particular, such a scheme must be resilient against *offline* password attacks. For PAPKE this means that an adversary who receives an authenticated public key and the ciphertexts created using this key cannot use offline computation to find the passwords used to create either object. In other words, the adversary cannot use the public key or the intercepted ciphertexts to locally test password guesses. Otherwise, the low entropy of passwords would hardly provide any extra security: according to NIST [14] even a 16-character human-memorizable password has only 30 bits of entropy on average, and hence can easily be brute-forced.

To illustrate this challenge, consider a few simple but failed attempts at constructing a secure PAPKE scheme. A natural way to password-authenticate any information, including a public key, would be to MAC it using the (hashed) password as a MAC key. This, however, would be subject to an offline dictionary attack, as the attacker can locally test password guesses until it finds the one for which the MAC verifies. More generally, any procedure which allows for *explicit verification* of the authenticated public key under a password would be subject to an offline attack. What if the key was not authenticated itself but the encrypting party included the password in the plaintext? This would be insecure against a man-in-the-middle attack which sends its public key to the encryptor and decrypts it to read the encryptor's password.

Indeed, in a secure PAPKE the authenticated public key must commit the receiver Alice to the password used in the key generation, and the sender Bob cannot verify this commitment explicitly, but it can create a ciphertext such that (1) it is correct if Bob's and Alice's passwords match, (2) the plaintext is

undecryptable if the two passwords differ, (3) the encryptor cannot tell which is the case, and (4) if the two passwords do not match then no one, including Alice who created the public key, can learn anything about Bob's password beyond the fact that it does not match the unique password she used to generate her public key.

We stress that PAPKE uses passwords to strictly enhance encryption security, i.e., for non-substituted keys, PAPKE provides standard CCA security that does not depend on the strength of the user's password. Thus the purpose of the password is solely to hedge security in case the encryptor uses a substituted key, but we stress that this hedging is applicable only if the encryptor shares a password with the party who generates the public key.

Our Contribution. We provide a thorough study of the proposed PAPKE primitive, and our contributions fall into the following three categories:

(Ia) Strong security notions for PAPKE. First, we formally introduce the concept of password-authenticated public key encryption, and define the desired security properties both via a universally composable (UC) functionality [17] and a set of property-based definitions. While property-based definitions are often more intuitive, a formalization in the UC framework provides stronger and more realistic security guarantees because it does not require any assumptions on the password distribution and correctly models real-world phenomena such as password reuse and making typing mistakes when entering the password. We prove that our UC security definition implies our property-based ones, hence proving a scheme secure in the UC setting implies its security under the more intuitive property-based notions.

(Ib) Relation to PAKE. To better understand the strength of the PAPKE primitive we compare it to the well-studied primitive of Password-Authenticated Key Exchange (PAKE) [8, 13, 18] The relation between PAPKE and PAKE is twofold. First, PAPKE immediately implies a two-round PAKE: Alice and Bob can perform password-authenticated key exchange if Alice sends to Bob a PAPKE public key authenticated by her password, and Bob encrypts a session key using the received key and his password. Indeed, we show that if this simple protocol is instantiated with any scheme satisfying our UC PAPKE notion then the resulting protocol satisfies the strong UC notion of PAKE [18].

Regarding the other direction it might seem at first glance that any 2-round PAKE protocol, e.g. [9, 8, 4], can generically imply a PAPKE scheme as follows: The PAKE requester's flow can define a PAPKE public key, and the PAKE responder's flow, together with an encryption of the plaintext under the established session key, can define a PAPKE ciphertext. However, we show that this intuition is in fact *incorrect*, as the non-interactive usage of encryption that is required by PAPKE is not compatible with standard PAKE security notions. This is because PAPKE must remain secure if the encryptor uses the same public key to encrypt multiple messages and the decryptor decrypts multiple ciphertexts using its private key, which here would be implemented by the temporary state of the PAKE requester. By contrast, PAKE requester flow is designed to be uti-

lized by a single honest responder session, and the requester session is designed to process only a single flow from the responder. Indeed, as we discuss below, the two-round PAKE implied by PAPKE has stronger security than what is implied by standard PAKE notions because it remains secure (and robust) even if either party re-uses its state. Summing up, the relation of PAPKE and PAKE is that PAPKE implies a 2-round PAKE with a (novel) property of security under session state re-use.

(II) Efficient PAPKE constructions. We show two very practical constructions that securely realize the UC PAPKE functionality. Our first construction generically builds a PAPKE scheme from a public-key encryption (PKE) scheme and an ideal cipher: The authenticated public key is an encryption of the PKE public key under the password, with the encryption implemented using an ideal cipher over the space of PKE public keys. To obtain the desired UC-security, the PKE scheme must satisfy a number of properties beyond standard CCA security, such as key-anonymity [6] and strong robustness [1]. We show a concrete instantiation of this scheme using a variant of DHIES [2] which satisfies these properties under the so-called Oracle Diffie-Hellman (ODH) assumption. This results in a highly-efficient construction secure under ODH in the Ideal Cipher model, which uses 1 exponentiation for key generation, 2 for encryption, and 1 for decryption.

However, ideal ciphers over arbitrary cyclic groups, e.g. an elliptic curve, are not so easy to implement. While generic constructions for ideal ciphers from random oracles exist [25, 19], implementing ideal ciphers over a specific algebraic group is not straightforward, and if not done carefully can result in timing and/or offline password guessing attacks. Thus we also provide an alternative concrete construction that does not rely on ideal ciphers and therefore might be easier to implement. It uses the Fujisaki-Okamoto transform [20] of a twisted "twin-key" ElGamal construction of independent interest. This construction uses 2 exponentiations for key generation, 2 multi-exponentiations for encryption, and 1 exponentiation and 1 multi-exponentiation in decryption, and relies on the Decisional Diffie-Hellman (DDH) assumption in the Random Oracle Model.

(IIIa) Efficient 2-Round UC PAKE schemes. Our generic PAPKE-to-PAKE compiler discussed above implies two highly efficient UC PAKE protocols when instantiated with the above two PAPKE schemes. To the best of our knowledge these are the first two-round UC-secure PAKE's which rely on standard cyclic groups, i.e., do not use groups with bilinear maps or other trapdoor structure, and which resort instead to either the Ideal Cipher (IC) or the Random Oracle Model (ROM) to achieve practical efficiency. Specifically, our results imply a UC PAKE which uses 2 expentiations per party but relies on an ideal cipher over a group, and a UC PAKE which uses 4 (multi)-exponentiations for the requester and 2 exponentiations for the responder and relies on a random oracle model for hash functions. Note that the first scheme matches and the second scheme comes very close to the 2 exponentiations/party cost of *unauthenicated* Diffie-Hellman Key Exchange, with is the minimum cost for PAKE one can reasonably expect. The closest efficiency-wise UC PAKE we know of is by Abdalla et al. [3], which

was shown secure under comparable assumptions, but which requires 3 message flows while our UC PAKE's use only 2 flows.

Beyond improving PAKE itself, a round-reduced PAKE yields round improvements in other protocols, e.g. the UC *asymmetric* PAKE (aPAKE) [21] for client-server authentication where the server keeps a hash of the client's password, can be realized by efficient compiler from UC PAKE [21, 27]. Shaving off a round in the underlying PAKE implies the same round-reduction in aPAKE's created by these generic compilers.

(IIIb) PAKE's with session re-use security. As we argued in (Ib) above, the PAPKE-to-PAKE compiler results in a 2-round PAKE which has novel security and reliability properties which follow from the fact that PAPKE enforces ciphertext security when the same public key is used to encrypt multiple messages. Recall that the PAKE requester message is a PAPKE public key, and the PAKE responder message is a PAPKE ciphertext encrypting a random session key under this public key, and both the public key and the ciphertext are created using the passwords of resp. the requester and the responder. (See Section 3.1 for the full description of this PAKE.) The *reliability* property of this PAKE is that the responder can re-use the requester message flow over multiple sessions, each of them generating a response, and for any (and all) of these responses that reaches the requester, the requester can use the same single session which generated the first message flow to generate multiple sub-sessions, each of which establishes a session key with the corresponding responder session. The *security* property is that each of these keys is secure even though they all re-used the single requester session state and the single requester message flow. The standard model of PAKE security does not guarantee security in this case, but a PAKE which is secure in this way can be beneficial to higher-level applications. For example it can help handle communication faults: A responder session which believes that its response has not been delivered correctly can safely respond to the same requester message again, and a requester who gets multiple responses can securely spin off a subprocess for each of them without re-starting a new session from scratch.

Roadmap. In Section 2 we define PAPKE as a strengthened version of public-key encryption, using both property-based definitions and an ideal functionality in the UC framework. Section 3 discusses the relation between PAKE and PAPKE, and shows a generic compiler building an efficient UC PAKE from any UC PAPKE, and an argument that the converse is not true, i.e., it is unlikely that PAKE implies PAPKE. Section 5 presents our two highly efficient UC PAPKE schemes, and Section 4 introduces the building blocks and assumptions needed in these constructions. In Section 6 we present the description of two highly-efficient concrete 2-round UC PAKE protocols obtained via our generic PAKE compiler of Section 3 applied to the PAPKE schemes of Section 5.

2 Security Model for PAPKE

In this section we introduce our security models for password-authenticated encryption. A peculiarity of formal security definitions for *password-based* primitives is that they must model the inherent probability of an adversary correctly guessing the low-entropy password. Property-based definitions [8] (sometimes also called game-based definitions) typically do so by requiring that the adversary's probability of winning the security game is negligibly more than a (non-negligible) threshold determined by its number of online queries and the entropy of the distribution from which the password is chosen. Composable security definitions [18] such as those in Canetti's Universal Composability (UC) framework [17], on the other hand, model the possibility of guessing the password directly into the ideal behavior of the primitive.

As argued by Canetti et al. [18], composable definitions provide stronger and more realistic security guarantees than property-based ones, because they do not make any implicit assumptions about the password distribution and correctly model real-world phenomena such as password reuse and typos while entering the password. Nevertheless, property-based definitions are often more intuitive and easier to understand than UC definitions. We therefore present both types of definitions: we first show the property-based PAPKE security notions in Section 2.1, and then the UC notion of PAPKE in Section 2.2. Finally, we prove that our UC security definition implies our property-based ones.

Syntax and High-Level Properties. Before we dive into our formal definitions, we define the syntax of PAPKE schemes and introduce their desired security properties. To this end, we first define the algorithms of a PAPKE scheme in the spirit of property-based definitions, i.e., there is no session identifier *sid* in the inputs to the algorithms, and the secret key is an output of key generation algorithm and an input to the decryption algorithm.

Definition 1 (PAPKE). *Let $\mathcal{D}$ be a dictionary of possible passwords, and $\mathcal{M}$ be a message space. A password-authenticated public-key encryption scheme is a tuple of algorithms* $\mathsf{PAPKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *with the following behavior:*

- $\mathsf{KGen}(\kappa, pwd) \rightarrow_R (apk, sk)$*: on input a security parameter κ and password $pwd \in \mathcal{D}$, output an authenticated public key apk and a secret key sk.*
- $\mathsf{Enc}(apk, pwd, m) \rightarrow_R c$*: on input an authenticated public key apk, password pwd and a message $m \in \mathcal{M}$, output a ciphertext c.*
- $\mathsf{Dec}(sk, c) \rightarrow m$*: on input a secret key sk and ciphertext c, output a message $m \in \mathcal{M} \cup \{\perp\}$ where $\perp$ indicates that the ciphertext is invalid.*

For correctness we require that for any password $pwd \in \mathcal{D}$, key pair $(apk, sk) \leftarrow_{\mathrm{R}} \mathsf{KGen}(\kappa, pwd)$, and ciphertexts $c \leftarrow_{\mathrm{R}} \mathsf{Enc}(apk, pwd, m)$, we have that $m = \mathsf{Dec}(sk, c)$.

Informally, the desired security properties of PAPKE schemes are:

Resistance against Offline-Attacks: None of the values that are (partially) derived from a password allows offline dictionary attacks on the passwords

that were used to generate them: The authenticated public key apk does not leak anything about the setup password pwd, and ciphertexts c formed under apk do not leak any information about the password attempt pwd' that was used in the encryption. The only and inevitable information leaked is that the party who holds the secret key sk corresponding to apk learns whether $pwd' = pwd$, because that holds if and only if $\mathsf{Dec}(sk, c) \neq \perp$.

CCA Security: Ciphertexts encrypted under an *honestly* generated authenticated public key apk hide the encrypted message from any adversary who doesn't know the secret key. This property is modeled in the standard CCA setting, and it holds even if the adversary knows all passwords used.

Security against Man-in-the-Middle (MITM) Attacks: The choice of an authenticated public key apk^* commits the adversary to some single password guess pwd^*, and all ciphertexts encrypted under apk^* using any password $pwd \neq pwd^*$ hide the encrypted message. The only available attack is an *online* attack, where the adversary guesses password pwd used by the honest encryptor and generates apk^* so that it commits to $pwd^* = pwd$. Thus the MITM attack gains effectively one password guess per each adversarial public key apk^* which the honest party uses in encryption.

Long-Term Security: The security of encryptions under an adversarially chosen key apk^* is preserved in a forward-secure manner because it holds even if the adversary (eventually) learns the encryptor's password $pwd \neq pwd^*$.

Ciphertext Authenticity: The password also guarantees authenticity of ciphertexts. That is, an adversary who knows an honestly generated key apk, but not the password pwd (or the secret key sk), cannot create valid ciphertexts, i.e., ciphertexts that decrypt under sk into some message $m \neq \perp$.

2.1 Property-Based Security Definition

We formalize the above intuitive security requirements using two game-based definitions, namely indistinguishability against chosen-ciphertext and chosen-key attack (IND-CCKA), and ciphertext authenticity (AUTH-CTXT). For the sake of brevity, we will refer to property IND-CCKA as the *privacy* property.

Privacy (IND-CCKA). Our notion of indistinguishability against chosen-ciphertext and chosen-key attacks (Def. 2) formalizes the first four properties sketched above. It can be seen as an extended version of the standard CCA notion that also incorporates active, man-in-the-middle attacks. We start by describing the passive CCA notion for our setting and then explain how defense against active attacks can be incorporated into it.

Passive Attacks. First, the adversary $\mathcal{A}$ receives the authenticated public key apk of a challenge key pair $(apk, sk) \leftarrow_{\mathrm{R}} \mathsf{KGen}(\kappa, pwd)$ that gets generated for a password pwd, chosen at random in some password space $\mathcal{D}$. The adversary is then given access to a "left-or-right" encryption oracle LoR which takes two messages m_0, m_1 and returns an encryption $C_b \leftarrow_{\mathrm{R}} \mathsf{Enc}(apk, pwd, m_b)$ for a

random choice of the bit b, and the task of the adversary is to determine this bit with probability non-negligibly better than random guessing. Equipped also with a decryption oracle for the corresponding secret key sk, this captures CCA security. We stress that this property does not rely on the password at all. In fact, the adversary is allowed to learn the password via a Reveal query.

Active Attacks. The standard CCA notion only guarantees indistinguishability for ciphertexts that are generated under the honest public key apk. PAPKE extends these guarantees to active attacks where the adversary tricks an honest encryptor into using his malicious public key apk^* instead of the real one. This is modeled in our IND-CCKA game by letting the adversary provide *any* public key apk^* as additional input to the LoR oracle. When the adversary provides a key $apk^* \neq apk$ to the LoR oracle he receives $C_b \leftarrow_{\mathrm{R}} \mathsf{Enc}(apk^*, pwd, m_b)$. Note that the password pwd used in the encryption by the LoR oracle is the "honest" one that was used to derive the challenge public key apk. As long as the adversarial key apk^* was not generated using the "honest" password, the ciphertext C_b should not leak any information about the encrypted message m_b.

Clearly, for such active attacks we must consider the possibility that the adversary correctly guesses the password pwd when generating apk^*, which allows him to decrypt the challenge ciphertext for apk^* and trivially win the security game. As every query to the LoR oracle where $apk^* \neq apk$ could be a potential password-guessing attack, we bound $\mathcal{A}$'s advantage in the formal definition by the number of such online guesses over the size of the password dictionary. The LoR oracle is not the only chance for $\mathcal{A}$ to try to guess the password though – if the adversary computes a ciphertext C^* using a guessed password pwd^* and the honest apk, then the decryption oracle will return $m \neq \perp$ only if $pwd^* = pwd$. Thus, the number of online password guesses considered in the active attack bound is the sum of $\mathcal{A}$'s decryption queries and the number of bad apk^*'s that $\mathcal{A}$ used towards the LoR oracle.

Online vs Offline Password Guessing. We stress that such online guessing attacks are unavoidable for active attacks and much less harmful than offline attacks against the password: every online attack requires the participation of an honest encryptor or decryptor, which naturally slows down the password guesses and allows an honest party to detect and throttle the attack. For instance, an honest user being notified that the public key of his friend changes on a daily basis will get suspicious and stop encrypting under these keys. Furthermore, the online guessing attacks only impact the achievable security for active attacks, but not for the passive attacks described above. This is modeled in our definition by checking if $q_{apk^*} = 0$, i.e., all LoR queries were made for the honest public key apk, in which case the experiment reduces to standard CCA security and the desired bound becomes independent of the number of $\mathcal{A}$'s oracle queries.

Long-Term Security. Finally, confidentiality of already generated ciphertexts should not be harmed by a subsequent exposure of the encryptor's password. We therefore grant $\mathcal{A}$ access to a Reveal oracle which will return the password

pwd that was used to derive *apk* and that is used in the LoR oracle. The Reveal oracle can be queried at any time in the game. However, after the adversary learns the password via a reveal query, he is no longer allowed to query the LoR oracle under malicious public keys $apk^* \neq apk$, as he could trivially win otherwise. He can still get challenge ciphertexts under the honest public key *apk* though, modeling that passive security does not rely on the password at all.

We describe this privacy experiment formally as follows:

Experiment $\mathsf{Exp}^{\mathsf{IND\text{-}CCKA}}_{\mathcal{A},\mathsf{PAPKE}}(\kappa)$:
$pwd \leftarrow_{\mathrm{R}} \mathcal{D}$, $\mathbf{L} \leftarrow \emptyset$, $(apk, sk) \leftarrow_{\mathrm{R}} \mathsf{KGen}(\kappa, pwd)$
$b \leftarrow_{\mathrm{R}} \{0,1\}$, $\texttt{revealed} \leftarrow 0$
$b' \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{LoR}(b,pwd,\cdot,\cdot,\cdot),\mathsf{Dec}(sk,\cdot),\mathsf{Reveal}(pwd)}(apk)$
oracle LoR on input a public key apk^* and
two messages m_0 and m_1 where $|m_0| = |m_1|$
if $apk^* \neq apk$ and $\texttt{revealed} = 1$, return $\perp$
else, compute $C \leftarrow_{\mathrm{R}} \mathsf{Enc}(apk^*, pwd, m_b)$,
if $apk^* = apk$ add C to $\mathbf{L}$
return C
oracle Dec on input a ciphertext $C \notin \mathbf{L}$:
return $m \leftarrow \mathsf{Dec}(sk, C)$ where $m \in \mathcal{M} \cup \{\perp\}$
oracle Reveal: return *pwd* and set $\texttt{revealed} \leftarrow 1$
return 1 if $b' = b$

Definition 2 (IND-CCKA)**.** *A* PAPKE *scheme is called* indistinguishable under chosen-ciphertext and key attacks *if for all efficient adversaries* $\mathcal{A}$*, and any password space* $\mathcal{D}$ *it holds that*

$$\Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CCKA}}_{\mathcal{A},\mathsf{PAPKE}}(\kappa) = 1] \leq \frac{1}{2} \;+\; \frac{1}{2} \cdot \frac{q_{apk^*} + q_{\mathsf{Dec}}}{|\mathcal{D}|} + \mathsf{negl}(\kappa)$$

for a negligible function negl*, where* q_{apk^*} *denotes the number of public keys* $apk^* \neq apk$ *that* $\mathcal{A}$ *used in its queries to the* LoR *oracle, and where:*

- *if* $q_{apk^*} > 0$*, then* q_{Dec} *is the number of* $\mathcal{A}$*'s queries to the* Dec *oracle while* $\texttt{revealed} = 0$ (active/MITM security)
- *if* $q_{apk^*} = 0$*, then* $q_{\mathsf{Dec}} \leftarrow 0$ (passive/CCA security)

In the IND-CCKA definition above we set $q_{\mathsf{Dec}} = 0$ for passive attacks, i.e. if $q_{apk^*} = 0$, then the security bound is $1/2 + \mathsf{negl}(\kappa)$. In other words, if $\mathcal{A}$ does not stage any MITM attack, i.e. it never substitutes the challenge public key *apk* with $apk^* \neq apk$, then IND-CCKA is like standard CCA-security of PKE, i.e. $\mathcal{A}$ can make any number of encryption and decryption queries and they will not impact its success probability.

Authenticity (AUTH-CTXT). The ciphertext authenticity property (Def. 3) formalizes that the adversary $\mathcal{A}$, given *apk* generated for password *pwd* chosen at random in dictionary $\mathcal{D}$, cannot create a valid ciphertext except for probability

$(1 + q_{apk^*} + q_{\mathsf{Dec}})/|\mathcal{D}|$, where q_{apk^*} is the number of encryption queries $\mathcal{A}$ makes under bad and distinct keys $apk^* \neq apk$, and q_{Dec} is the number of decryption queries. The reason that each decryption query and each encryption query for $apk^* \neq apk$ can both serve as password-guessing oracles are explained in the privacy game discussion above. Note that here we do not ever let $\mathcal{A}$ learn pwd because knowing pwd suffices to form a valid ciphertext. The additional constant 1 in the password-guessing count $1 + q_{apk^*} + q_{\mathsf{Dec}}$ is included because the final ciphertext $\mathcal{A}$ creates, can itself be used to guess a password.

The authenticity experiment is defined as follows:

Experiment $\mathsf{Exp}^{\mathsf{AUTH\text{-}CTXT}}_{\mathcal{A},\mathsf{PAPKE}}(\kappa)$:
$pwd \leftarrow_{\mathrm{R}} \mathcal{D}$, $\mathbf{L} \leftarrow \emptyset$, $(apk, sk) \leftarrow_{\mathrm{R}} \mathsf{KGen}(\kappa, pwd)$
$C^* \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Enc}(pwd,\cdot,\cdot),\mathsf{Dec}(sk,\cdot)}(apk)$
oracle Enc on input a key apk^* and message m :
compute $C \leftarrow_{\mathrm{R}} \mathsf{Enc}(apk^*, pwd, m)$
if $apk^* = apk$ add C to $\mathbf{L}$
return C
oracle Dec on input a ciphertext C :
return $m \leftarrow \mathsf{Dec}(sk, C)$, where $m \in \mathcal{M} \cup \{\perp\}$
return 1 if $\mathsf{Dec}(sk, C^*) \neq \perp$ and $C^* \notin \mathbf{L}$

Definition 3 (AUTH-CTXT). *A* PAPKE *scheme provides* authenticity of ciphertexts *if for all efficient adversaries $\mathcal{A}$, and any password space $\mathcal{D}$ it holds that*

$$\Pr[\mathsf{Exp}^{\mathsf{AUTH\text{-}CTXT}}_{\mathcal{A},\mathsf{PAPKE}}(\kappa) = 1] \leq \frac{q_{apk^*} + q_{\mathsf{Dec}} + 1}{|\mathcal{D}|} + \mathsf{negl}(\kappa)$$

for a negligible function negl*, where q_{apk^*} is the number of bad keys $apk^* \neq apk$ that $\mathcal{A}$ used in queries to the* Enc *oracle and q_{Dec} is the number of queries to the* Dec *oracle.*

2.2 Security Definition of PAPKE in the UC Framework

It is well-known that property-based definitions have limitations when it comes to capturing natural password distributions and usages: they assume that honest users choose their passwords at random from known, fixed, independent distributions, whereas in reality users tend to share or reuse passwords and also leak information related to their passwords.

We therefore also formalize PAPKE security in the Universal Composability (UC) framework [17] which models passwords more naturally. In UC, one defines security by describing an ideal functionality $\mathcal{F}$ that performs the desired task in a way that is secure-by-design. A real-world cryptographic scheme is then said to securely realize a certain ideal functionality $\mathcal{F}$, if an environment $\mathcal{Z}$ cannot distinguish whether it is interacting with the real scheme or with $\mathcal{F}$ and a simulator SIM. The crux is that all inputs to honest parties, such as their passwords, are provided by the environment which avoids assumptions regarding the distributions, dependencies, or leakages of user passwords.

1. `Key Generation.` On input $(\mathsf{KEYGEN}, sid, pwd)$ from party $\mathcal{P}$:
- If $sid \neq (\mathcal{P}, sid')$ or a record $(\mathsf{keyrec}, sid, \cdot, \cdot)$ exists, ignore.
- Send (KEYGEN, sid) to $\mathcal{A}$ and wait for $(\mathsf{KEYCONF}, sid, apk, \mathcal{M})$ from $\mathcal{A}$.
- If a record $(\mathsf{badkeys}, sid, apk_j, \cdot)$ with $apk_j = apk$ exists, abort.
- Create record $(\mathsf{keyrec}, sid, apk, pwd)$ and output $(\mathsf{KEYCONF}, sid, apk)$ to $\mathcal{P}$.

2. `Encryption.` On input $(\mathsf{ENCRYPT}, sid, apk', pwd', m)$ from party $\mathcal{Q}$ where $m \in \mathcal{M}$:
- If a record $(\mathsf{keyrec}, sid, apk, pwd)$ with $apk = apk'$ exists and $\mathcal{P}$ (from $sid = (\mathcal{P}, sid')$) is honest, then do the following:
  - Send $(\mathsf{ENC\text{-}L}, sid, |m|)$ to $\mathcal{A}$ and wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
  - Abort if a record $(\mathsf{encrec}, sid, \cdot, c)$ for c already exists.
  - Create record $(\mathsf{encrec}, sid, m', c)$ with $m' \leftarrow m$ if $pwd' = pwd$, and $m' \leftarrow \perp$ else.
- Otherwise do the following:
  - If a record $(\mathsf{badkeys}, sid, apk_j, pwd_j)$ with $apk_j = apk'$ exists, set $pwd^* \leftarrow pwd_j$.
  - Else, send $(\mathsf{GUESS}, sid, apk')$ to $\mathcal{A}$, wait for $(\mathsf{GUESS}, sid, pwd^*)$ from $\mathcal{A}$ and create record $(\mathsf{badkeys}, sid, apk', pwd^*)$.
  - If $pwd' = pwd^*$, send $(\mathsf{ENC\text{-}M}, sid, m)$ to $\mathcal{A}$, wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
  - If $pwd' \neq pwd^*$, send $(\mathsf{ENC\text{-}L}, sid, |m|))$ to $\mathcal{A}$, wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
- Output $(\mathsf{CIPHERTEXT}, sid, c)$ to $\mathcal{Q}$.

3. `Decryption.` On input $(\mathsf{DECRYPT}, sid, c)$ from party $\mathcal{P}$:
- If $sid \neq (\mathcal{P}, sid')$ or no record $(\mathsf{keyrec}, sid, apk, pwd)$ exists, ignore.
- If a record $(\mathsf{encrec}, sid, m, c)$ for c exists, where $m \in \mathcal{M} \cup \{\perp\}$:
  - Output $(\mathsf{PLAINTEXT}, sid, m)$ to $\mathcal{P}$.
- Else (i.e., if no such record exists):
  - Send $(\mathsf{DECRYPT}, sid, c)$ to $\mathcal{A}$ and wait for $(\mathsf{PLAINTEXT}, sid, m, pwd^*)$ from $\mathcal{A}$.
  - If $pwd^* = pwd$, set $m' \leftarrow m$ and $m' \leftarrow \perp$ otherwise.
  - Create record $(\mathsf{encrec}, sid, m', c)$.
  - Output $(\mathsf{PLAINTEXT}, sid, m')$ to $\mathcal{P}$.

Fig. 1. Password-Authenticated PKE (PAPKE) functionality $\mathcal{F}_{\mathsf{PAPKE}}$ parametrized with message space $\mathcal{M}$.

Ideal Functionality $\mathcal{F}_{\mathsf{PAPKE}}$. We define our functionality $\mathcal{F}_{\mathsf{PAPKE}}$, Figure 1, that models the ideal behavior of a PAPKE scheme in the UC framework. $\mathcal{F}_{\mathsf{PAPKE}}$ is an extension of the functionality $\mathcal{F}_{\mathsf{PKE}}$ for standard public-key encryption [17, 15], and improves the confidentiality and authenticity guarantees by leveraging a (shared) password. For ease of comparison between $\mathcal{F}_{\mathsf{PKE}}$ and $\mathcal{F}_{\mathsf{PAPKE}}$, we present the standard functionality $\mathcal{F}_{\mathsf{PKE}}$ adapted to our notation in Appendix B. We assume static corruptions in our paper, which is the standard in the context of PKE for which PAPKE is an extension of. Considering adaptive corruptions for PKE hardly increase practical security but requires non-committing ciphertexts which is not achievable in the standard model [33]. We also stress that our model protects against adaptive corruption of *passwords*, which is a more important threat: PAPKE ensures forward security of past encryptions under malicious keys, even if the password used in encryption is subsequently leaked.

In the following we discuss the intuition behind functionality $\mathcal{F}_{\mathsf{PAPKE}}$ and explain how it enforces the desired PAPKE security properties.

Interfaces and Parties. The functionality provides three interfaces: KEYGEN, ENCRYPT and DECRYPT which resemble the PAPKE algorithms introduced before. As required by the UC framework, all interfaces get a globally unique session identifier *sid* as input, which allows us to focus the analysis on a single scheme instance. We use session identifiers of the form $sid = (\mathcal{P}, sid')$ for some party $\mathcal{P}$ and a unique string sid'. The KEYGEN and DECRYPT interface are only accessible to the party $\mathcal{P}$ that is specified in the *sid*, whereas the ENCRYPT interface can be invoked by any party $\mathcal{Q}$. This reflects the public-key setting: everyone can encrypt, but only the designated receiver of a message can decrypt. The high-level idea is to model secure encryption by letting the functionality only output dummy ciphertexts c that are independent of the plaintext m. Decryption is realized by keeping internal records that link these dummy ciphertexts to the encrypted messages.

Security Against Active & Passive Attacks. As in $\mathcal{F}_{\mathsf{PKE}}$, the ideal functionality for public key encryption, we let the adversary $\mathcal{A}$ provide the dummy ciphertexts c, but extend the cases where he has to do this blindly, i.e., without learning the plaintext. In $\mathcal{F}_{\mathsf{PKE}}$, the adversary learns the plaintext whenever a party $\mathcal{Q}$ calls the ENCRYPT interface for a wrong key $pk' \neq pk$, i.e., not the key pk that was registered by the KEYGEN interface. In our $\mathcal{F}_{\mathsf{PAPKE}}$ definition this is no longer the case, and $\mathcal{A}$ only receives the plaintext when a message is encrypted under a wrong key $apk' \neq apk$ *and* $\mathcal{A}$ correctly guesses the encryption password pwd' used by the encryptor $\mathcal{Q}$.

If $\mathcal{A}$'s password guess is incorrect, he does not receive the message but only the length of the plaintext. Thus, if $\mathcal{A}$ doesn't know the password, $\mathcal{F}_{\mathsf{PAPKE}}$ treats encryption under a malicious apk' the same way as encryption under the correct and "honest" public key. As the adversary has to provide c without having learned m, it follows that c cannot leak information about m either. Note that we only give the adversary a single password guess per bad apk' and also ensure that a bad apk' can never become a good one (via the KEYGEN interface).

Authenticity. The functionality $\mathcal{F}_{\mathsf{PAPKE}}$ realizes decryption via internal records $(\mathsf{encrec}, sid, m, c)$ that it creates whenever the ENCRYPT interface is invoked on the correct key apk. Upon a DECRYPT call for some ciphertext c, $\mathcal{F}_{\mathsf{PAPKE}}$ either retrieves m from the record or, if no record exists, asks the adversary to provide a plaintext.

In contrast to $\mathcal{F}_{\mathsf{PKE}}$, our functionality $\mathcal{F}_{\mathsf{PAPKE}}$ guarantees that decryption can only succeed if the ciphertext is generated using the correct password. That is, if an ENCRYPT call is made for an incorrect password, then an internal record gets still created, but with $m \leftarrow \perp$. Likewise, for any decryption of a ciphertext c for which no record exists, i.e., c was created outside of the functionality, the adversary is asked to provide a password guess pwd^* along with the plaintext m. Only if the password guess matches the setup password, this message gets returned to $\mathcal{P}$.

Modeling Password Compromise. Finally, we stress that our $\mathcal{F}_{\mathsf{PAPKE}}$ definition captures password compromise, even though it does not show up in a dedicated interface: In the UC framework, inputs to honest participants are provided by the environment. That is, the environment is always privy of the setup password of the honest users, as it chooses them. The environment can pass arbitrary information to the adversary, including the password, so there is no need to provide an additional reveal interface. $\mathcal{F}_{\mathsf{PAPKE}}$ defines the impact of knowing or guessing the correct password when encrypting messages.

$\mathcal{F}_{\mathsf{PAPKE}}$ $\Rightarrow$ IND-CCKA + AUTH-CTXT. Our ideal functionality $\mathcal{F}_{\mathsf{PAPKE}}$ implies our property-based definitions, in the sense that any scheme $\Pi_{\mathsf{PAPKE}} = (\mathsf{KEYGEN}, \mathsf{ENCRYPT}, \mathsf{DECRYPT})$ that securely implements the $\mathcal{F}_{\mathsf{PAPKE}}$ functionality in the UC framework is also IND-CCKA and AUTH-CTXT secure (where these definitions are adapted to the syntax of the UC algorithms KEYGEN, ENCRYPT, and DECRYPT). Therefore our proofs of UC security in later sections imply security under the property-based notions.

Theorem 1. *Any* PAPKE *scheme* Π_{PAPKE} *that securely realizes* $\mathcal{F}_{\mathsf{PAPKE}}$ *is also* IND-CCKA *and* AUTH-CTXT *secure.*

The proof of the above theorem is deferred to Appendix A.

3 Relation between PAPKE and PAKE

PAPKE, the new cryptographic primitive we propose, is closely related to Password Authenticated Key Agreement (PAKE) [8, 13, 18]. Specifically, we show that it is easy to build a (UC-secure) two-round PAKE scheme from a (UC-secure) PAPKE scheme, but that while the converse looks like it should be true at first sight, it is not true in general, because PAPKE has stricter properties than a standard PAKE. In particular, we give a counterexample of a secure two-round PAKE scheme that, when converted into a PAPKE scheme in the straightforward fashion, yields an insecure PAPKE scheme. Indeed, PAPKE can be thought of as a two-round PAKE with a novel property of security under session state re-use, which to the best of our knowledge has not been observed and provably realized before.

3.1 Constructing PAKE from PAPKE

We show that any UC-secure PAPKE can be converted into a two-round UC-secure PAKE. (We include the standard UC PAKE security notion defined via an ideal functionality $\mathcal{F}_{\mathsf{PAKE}}$ [18] in Figure 10 in Appendix B.) This construction is shown in Figure 2, and it is fairly simple: The initiator $\mathcal{P}_i$ generates an authenticated public key apk from the input password pwd and sends it to $\mathcal{P}_j$. The responder $\mathcal{P}_j$, given its password pwd' and the received public key apk, picks a random session key $k \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$, and responds to $\mathcal{P}_i$ with an encryption

of k under apk and pwd'. $\mathcal{P}_i$ receives key k by decrypting the received ciphertext, or outputs $\perp$ if the decryption fails. Note that all communication is done over an insecure channel, fully controlled by the adversary. In particular, an adversary can replace $\mathcal{P}_i$'s public key and/or $\mathcal{P}_j$'s ciphertext. However, PAPKE security implies that neither $\mathcal{P}_i$'s public key or $\mathcal{P}_j$'s ciphertext reveal anything about passwords, resp. pwd and pwd', and the only attack the adversary can stage is an on-line guessing attack, because each substituted public key apk^* or ciphertext c^* commits the adversary to a *single* password guess pwd^*, and is guaranteed to fail (e.g. $\mathcal{P}_j$ fails to encrypt anything useful under apk^* or $\mathcal{P}_i$ fails to decrypt c^*) unless the guessed password pwd^* matches the password of resp. $\mathcal{P}_j$ or $\mathcal{P}_i$.

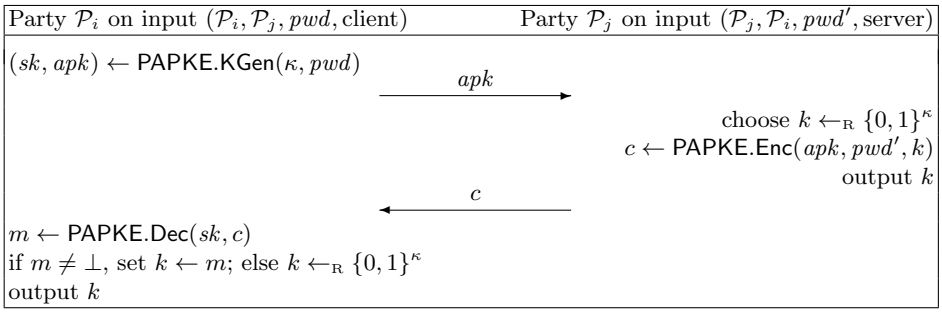


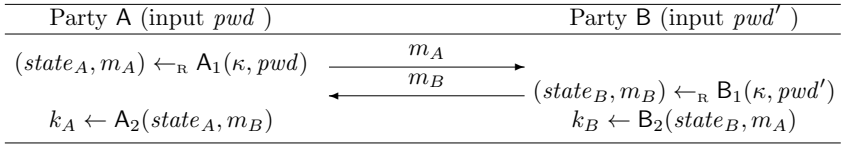
Fig. 2. Two-round PAKE protocol PAPKE-2-PAKE assuming a UC-secure PAPKE = (KGen, Enc, Dec). Note that the messages are sent over an insecure channel, i.e., an adversary is able to see and modify both apk and c.

The proof of the following theorem is deferred to Appendix C:

Theorem 2. *If* PAPKE *realizes the UC PAPKE functionality* $\mathcal{F}_{\mathsf{PAPKE}}$ *of Figure 1, Section 2.2), then the* PAPKE-2-PAKE *scheme shown in Figure 2 realizes the UC PAKE functionality* $\mathcal{F}_{\mathsf{PAKE}}$ *shown in Figure 10, Appendix B.*

3.2 An intuitive PAKE-2-PAPKE compiler, and why it doesn't work

It turns out that the intuitive approach of building PAPKE from two-round PAKE does not work due to subtle differences in the security notions of both primitives. Indeed, PAPKE has some security properties which are stronger than PAKE, and this in particular implies that the PAPKE-to-PAKE compiler of Section 3.1 adds a new security property to the resulting PAKE. (We discuss that PAKE security property in Section 3.3 below.) For the ease of exposition, we state our results for the game-based representations of PAKE and PAPKE instead of using their UC variants, and refer to parties $\mathcal{P}_i$ and $\mathcal{P}_j$ as A and B respectively. On a first glance, it seems reasonable to generically build a PAPKE scheme from any two-round PAKE protocol, e.g. [9, 8, 4]. Specifically, any two-round PAKE protocol $\langle(\mathsf{A}_1, \mathsf{A}_2) \rightleftharpoons (\mathsf{B}_1, \mathsf{B}_2)\rangle$ can be abstracted as follows:



The natural approach to constructing PAPKE would combine a two-round PAKE with an authenticated encryption scheme AE: The PAKE message m_A from A would be A's static authenticated public key apk, and to encrypt message m under A's key $apk = m_A$ any party could complete the two-round PAKE protocol in the role of B and append the AE encryption of m under the derived session key k_B to B's PAKE message m_B. For decryption, A uses m_B to complete her side of the PAKE protocol to derive the same session key $k_A = k_B$ (if $pwd = pwd'$) and uses k_A to decrypt the attached ciphertext. More formally, given a 2-round $\mathsf{PAKE} = \langle(\mathsf{A}_1, \mathsf{A}_2) \rightleftharpoons (\mathsf{B}_1, \mathsf{B}_2)\rangle$ and authenticated encryption $\mathsf{AE} = (\mathsf{AE.Enc}, \mathsf{AE.Dec})$ sharing the same key space $\mathcal{K}$, one could consider the following PAPKE construction:

$\mathsf{PAPKE.KGen}(\kappa, pwd)$:
  run $(state_A, m_A) \leftarrow_{\mathrm{R}} \mathsf{A}_1(\kappa, pwd)$, return $(sk \leftarrow state_A, apk \leftarrow m_A)$
$\mathsf{PAPKE.Enc}(apk, pwd', m)$:
  run $(state_B, m_B) \leftarrow_{\mathrm{R}} \mathsf{B}_1(\kappa, pwd')$ and $k_B \leftarrow \mathsf{B}_2(state_B, apk)$
  encrypt $c \leftarrow \mathsf{AE.Enc}(k_B, m)$ and return $c' \leftarrow (m_B, c)$
$\mathsf{PAPKE.Dec}(sk, c')$:
  parse $c' = (m_B, c)$ and $sk = state_A$
  get $k_A \leftarrow \mathsf{A}_2(state_A, m_B)$ and return $m \leftarrow \mathsf{AE.Dec}(k_A, c)$

Intuitively, this should yield a secure PAPKE if PAKE is secure and AE is a secure authenticated encryption scheme. However, this generic construction uses PAKE in a way that is not covered by its security definition: Whenever party A decrypts a PAPKE ciphertext it effectively *re-uses* the same local PAKE session state $state_A$ (and the same first-round message m_A) across multiple PAKE sessions. Indeed, this gap can be exploited to craft special PAKE and AE schemes that are secure by themselves but result in an insecure PAPKE when used in this natural compiler.

Counterexample: $\mathsf{PAKE} + \mathsf{AE} \not\Rightarrow \mathsf{PAPKE}$. The basic idea is that we extend a secure PAKE scheme such that it allows a malicious party to recover A's input password pwd when A reuses the same message m_A (or rather internal state $state_A$) multiple times in different PAKE executions. For the sake of simplicity we assume that pwd is encoded as a κ-bit long string.

PAKE' : Given a secure scheme PAKE (and a secure $\mathsf{PRG} : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$) we derive the scheme PAKE' described in Figure 3. In our modified scheme we steer A's behavior with two bits $b = b_1 b_2$ that get prepended to B's round message m_B. An honest party B will always use $b = 00$ for which A will normally execute the PAKE protocol and both parties extend their key via a PRG. A malicious

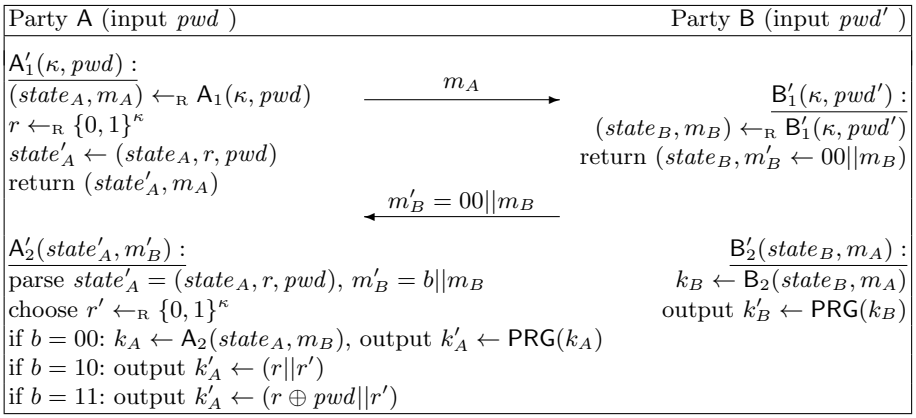


Fig. 3. PAKE' construction for our counterexample.

user B can set $b_1 = 1$ that invokes A in a "bad" mode in which A either returns or $(r||r')$ or $(r \oplus pwd||r')$ as session key. While r' is chosen fresh for every second round message, r is a random value that gets chosen at the beginning of every PAKE session and becomes part of A's state $state_A$. In the standard PAKE context, where a malicious B can run only a single session with A using the same state r, our modified PAKE' scheme is as secure as PAKE. However, it loses all security if an adversary can force an honest party to reuse its state $state_A$ (and thus r) in multiple sessions, as it then learns pwd.

AE' : We also change a secure authenticated encryption scheme $\mathsf{AE} = (\mathsf{AE.Enc}, \mathsf{AE.Dec})$ into the following version AE' with keys whose length is $2 \cdot \kappa$.

$\mathsf{AE}'.\mathsf{Enc}(k, m)$: parse $k = k_L||k_R$ and return $c' \leftarrow (0, c)$ where $c \leftarrow_{\mathrm{R}} \mathsf{AE.Enc}(k_R, m)$
$\mathsf{AE}'.\mathsf{Dec}(k, c)$: parse $k = k_L||k_R$ and $c' = (b, c)$,
if $b = 0$ then return $m \leftarrow \mathsf{AE.Dec}(k_R, c)$; else return k_L

When used "correctly", AE' will always use the right half of the key and behave exactly as AE. Only when decryption is invoked on special ciphertexts, it will reveal the left part of the key (which in combination with PAKE' will be the masked password or the random mask). Clearly, AE' is still a secure encryption scheme, as the left halve of the key never gets used.

While PAKE' and AE' are secure building blocks by itself, the derived PAPKE scheme would not satisfy our $\mathsf{IND\text{-}CCKA}$ notion. An adversary simply makes two decryption queries for $m'_B = 10\|m_B$ and $m'_B = 11\|m_B$ upon which he learns both r and $r \oplus pwd$ and can recover the password pwd.

3.3 Implications for UC PAKE protocols

We discuss the main conclusions we draw from the two technical facts above.

First 2-round UC PAKE's competitive with game-based PAKE's. In Section 6.2 we include two highly efficient UC PAKE protocols by instantiating

the PAPKE-2-PAKE compiler with the PAPKE constructions of Section 5. To the best of our knowledge these are the first 2-round UC PAKE's which rely on standard cyclic groups with efficiency comparable to the Diffie-Hellman key exchange in the IC or RO model. While UC PAKE can be achieved using even 1 (simultaneous) round of communication, all 1-round UC PAKEs we know, e.g. [32, 31], use groups with bilinear maps and are significantly costlier. Thus practitioners are likely to resort to constructions which require IC or ROM models but give much better concrete efficiency.

Concretely, we show two 2-round UC PAKE protocols: PAKE-IC-DHIES, secure under Oracle Diffie-Hellman (ODH) assumption (see Section 4) in the IC model which uses 2 exponentiations per party, and PAKE-FO, secure under DDH in ROM which uses 4 (multi-)exps for the requester and 2 for the responder. These schemes *almost* match the efficiency and assumptions used by 2-round PAKE's that satisfy only *game-based PAKE* security notions [8, 13, 4], e.g. 2 exps per party under DDH in ROM in [4]. By contrast, the previously known UC PAKE shown secure under comparable assumptions, specifically DDH in IC model, by Abdalla et al. [3], requires 3 rounds, while our UC PAKE's use only 2 rounds, thus matching the round complexity of efficient game-based PAKE's.

The Universally Composable notion of PAKE security [18] has long been recognized as stronger than the game-based notions [8, 13], not only because it implies concurrent security and can be used in protocol composition, but also because, unlike the game-based notions, the UC PAKE implies security for non-uniform password distributions, password re-use, correlated passwords, misstyped passwords, and any other forms of information leakage. However, there has been an efficiency and round-complexity gap between UC PAKE's and PAKE's shown secure only under game-based notions, with the 3-round 2-exp/party UC PAKE of Abdalla et al. [3] coming closest to the 2-round 2-exp/party game-based PAKE's, e.g. [8, 13, 4]. Bridging this gap allows practitioners to adopt solutions which provably achieve these strong security properties, i.e. a UC PAKE, without paying a penalty in computation or round complexity.

PAKE with security on session re-use. As we argued in section 3.2 above, the reason the compiler from 2-round PAKE to PAPKE does not work is that a standard PAKE security model does not extend to the case of the requester party, A, re-using the local state $state_A$ of a single PAKE session across many sessions, each of which would derive a session key k_A from same state k_A but potentially different responder messages m_B. By contrast, PAKE created from the secure PAPKE in Figure 2 does have this property: The requester party $\mathcal{P}_i$ can use the same local state, which is the PAPKE secret key sk, across many sessions, deriving $k_A \leftarrow \mathsf{PAPKE.Dec}(sk, c)$ on any number of responder messages c. By the same token, the responder $\mathcal{P}_j$ in this PAKE protocol is free to re-use $\mathcal{P}_i$'s first-round message apk in multiple sessions, because PAPKE ciphertexts created in each such session are all secure, and their plaintexts can all be used as session keys. Note that we do not formally model this "session state re-use" PAKE security property, but the security of this usage of PAPKE follows immediately

from PAPKE security definition, because CCA security of PAPKE implies that the public key pair (sk, apk) can be used to securely encrypt multiple messages.

Indeed, this shows that protocol PAPKE-2-PAKE is a secure 2-round PAKE with security under re-use of requester's session state across multiple sessions. This novel security property can simplify PAKE implementations in practice because it means that it is safe for the responder to re-use the requester message flow over several sessions, and it is safe for the requester to use the same session state to process each of such responses. This makes it easier for the higher-level application to handle faults, because both parties can keep their session information, the session state $state_A = sk$ for the requester and the requester's first message $m_A = apk$ for the responder, and re-use them in case of communication faults instead of re-starting a new session from scratch.

Applications to Higher-Level Protocols. The 2-round UC PAKE's yield round improvements for higher-level protocols that rely on UC PAKE. For example, in the client-server setting for password authentication the server does not hold the client's password itself, but only password hashes, so that server compromise can reveal passwords only via an explicit offline dictionary attack. This setting, known as *asymmetric*, a.k.a. *augmented* or *verifier-based*, PAKE (aPAKE), was formalized first in the game-based approach by Boyko et al. [13] and then in the UC framework by Gentry et al. [21]. Known UC aPAKE's include two PAKE-to-aPAKE compilers of [21, 27] which add 1 exp/party and resp. 2 or 1 message flows to UC PAKE, the 1 round (of simultaneous communication) aPAKE of [30] which relies on bilinear maps and is significantly more expensive, and a 3-round and 3 - 4 exp/party aPAKE scheme *OPAQUE* of [28] based on the One-More Diffie-Hellman (OM-DH) assumption.[5] Using our 2-round PAKE's in the compiler of [27] gives 3-round aPAKE's which use resp. 5 - 3 exp/party based on DDH or 3 exp's based on ODH. We note that [28] constructs a *strong* aPAKE, which strengthens aPAKE's by eliminating the benefits of precomputation in an offline dictionary attack, while the aPAKE implied by PAKE-FO and the compiler of [27] would not have that property. On the other hand, the aPAKE construction of [28] requires an interactive OM-DH assumption while the aPAKE implied by PAKE-FO and the compiler of [27] requires only DDH.

4 Building Blocks in PAPKE Constructions

We include the building blocks and assumptions needed in the PAPKE constructions of Section 5. The first construction relies on generic public-key encryption with additional key-anonymity and robustness properties, while the second construction is based on the Decisional Diffie-Hellman assumption.

Public-Key Encryption. We recall the standard notion of public-key encryption scheme, $\mathsf{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, which consists of a key generation al-

[5] In the original publication [29] OPAQUE was presented as a 2-round protocol, but its security as a realization of the UC aPAKE functionality seems to require a third round with an explicit client-to-server authentication.

gorithm that on input a security parameter outputs a key pair $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{PKE.KGen}(\kappa)$, an encryption algorithm that on input the public key and a message outputs a ciphertext $c \leftarrow_{\mathrm{R}} \mathsf{PKE.Enc}(pk, m)$, and a decryption algorithm that on input the secret key and a ciphertext outputs a message $m \leftarrow \mathsf{PKE.Dec}(sk, c)$ or outputs $\perp$ to indicate that the ciphertext is invalid. Correctness requires that $\mathsf{PKE.Dec}(sk, \mathsf{PKE.Enc}(pk, m)) = m$ with probability one for all m and $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{PKE.KGen}(\kappa)$.

Uniform Public-Key Space. We say that PKE has *uniform public-key space* $\mathcal{PK}$ if the public key output of $\mathsf{PKE.KGen}(\kappa)$ induces a uniform distribution over set $\mathcal{PK}$. All PKE schemes in a prime-order group setting have this property, e.g. their public keys are uniformly chosen group elements.

Anonymity & Indistinguishability (AI-CCA). In addition to standard CCA security, we also need the PKE scheme to be key anonymous, meaning that a ciphertext cannot be linked to the public key under which it is encrypted. We use the AI-CCA notion of [12, 1], which combines indistinguishability [22, 7] and anonymity [6], both under CCA attacks.

Definition 4 (AI-CCA). *A public-key encryption scheme* PKE = (PKE.Enc, PKE.Dec) *is* AI-CCA*-secure if for all efficient adversaries* $\mathcal{A}$*, it holds that* $|\Pr[\mathsf{Exp}^{\mathsf{AI\text{-}CCA}}_{\mathcal{A},\mathsf{PKE}}(\kappa) = 1] - 1/2| \leq \mathsf{negl}(\kappa)$ *for some negligible function* negl.

Experiment $\mathsf{Exp}^{\mathsf{AI\text{-}CCA}}_{\mathcal{A},\mathsf{PKE}}(\kappa)$:
$(pk_0, sk_0) \leftarrow_{\mathrm{R}} \mathsf{PKE.KGen}(\kappa)$, $(pk_1, sk_1) \leftarrow_{\mathrm{R}} \mathsf{PKE.KGen}(\kappa)$
$b \leftarrow_{\mathrm{R}} \{0, 1\}$
$b' \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{LoR}(b,\cdot,\cdot),\mathsf{Dec}(\cdot,\cdot)}(\kappa, pk_0, pk_1)$ where
  oracle LoR can be queried only once on input two messages m_0, m_1 with $|m_0| = |m_1|$ to return $c^* \leftarrow_{\mathrm{R}} \mathsf{PKE.Enc}(pk_b, m_b)$
  oracle Dec on input $d \in \{0, 1\}$ and a ciphertext $c \neq c^*$:
    return $m \leftarrow \mathsf{PKE.Dec}(sk_d, c)$
return 1 if $b' = b$

Strong Robustness (SROB-CCA). We also need the robustness property of PKE, which enforces that an adversary cannot create a ciphertext which is valid under two different keys. We use the notion of strong robustness under chosen ciphertext attacks, SROB-CCA, by Abdalla et al. [1], except we restrict it to PKE schemes instead of considering general encryption schemes that encompass both PKE and identity-based encryption.

Definition 5 (SROB-CCA). *A public-key encryption scheme* PKE = (PKE.Enc, PKE.Dec) *is* SROB-CCA*-secure if for all efficient adversaries* $\mathcal{A}$*, it holds that* $|\Pr[\mathsf{Exp}^{\mathsf{SROB\text{-}CCA}}_{\mathcal{A},\mathsf{PKE}}(\kappa) = 1]| \leq \mathsf{negl}(\kappa)$ *for some negligible function* negl.

Experiment $\mathsf{Exp}^{\mathsf{SROB\text{-}CCA}}_{\mathcal{A},\mathsf{PKE}}(\kappa)$:
$(pk_0, sk_0) \leftarrow_{\mathrm{R}} \mathsf{PKE.KGen}(\kappa)$, $(pk_1, sk_1) \leftarrow_{\mathrm{R}} \mathsf{PKE.KGen}(\kappa)$
$c \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Dec}(\cdot,\cdot)}(\kappa, pk_0, pk_1)$ where
  oracle Dec on input $d \in \{0,1\}$ and a ciphertext c :
    return $m \leftarrow \mathsf{PKE.Dec}(sk_d, c)$
$m_0 \leftarrow \mathsf{PKE.Dec}(sk_0, c)$, $m_1 \leftarrow \mathsf{PKE.Dec}(sk_1, c)$
return 1 if $m_0 \neq \perp$ and $m_1 \neq \perp$

Secure Instantiation. Abdalla et al. [1] show that the DHIES scheme satisfies the SROB-CCA and AI-CCA notion when a minor modification is made to exclude zero randomness and rejects ciphertexts that have 1 as first component. We recall their DHIES* scheme of [1] in Section 6.

Diffie-Hellman Assumptions. Our second construction requires a group $(\mathbb{G}, g, p)$ as input where $\mathbb{G}$ denotes a cyclic group $\mathbb{G} = \langle g \rangle$ of prime order p in which the *Decisional Diffie-Hellman (DDH)* problem is hard with respect to a security parameter κ, i.e., p is a κ-bit prime. A group $(\mathbb{G}, g, p)$ satisfies the DDH assumption if for all efficient adversaries $\mathcal{A}$, $\big|\Pr[\mathcal{A}(\mathbb{G}, p, g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(\mathbb{G}, p, g, g^a, g^b, g^c) = 1]\big|$ is negligible in κ, where the probability is over the random choices of p, g, the random choices of $a, b, c \in \mathbb{Z}_p$, and $\mathcal{A}$'s coin tosses.

In some game hops of our proof, we only need the weaker assumption of the *Computational Diffie-Hellman (CDH)* problem. Using the notation from above, a group $(\mathbb{G}, g, p)$ satisfies the CDH assumption if for all efficient adversaries $\mathcal{A}$, the probability $\Pr[\mathcal{A}(\mathbb{G}, p, g, g^a, g^b) = g^{ab}]$ is negligible in κ, where the probability is over the random choices of p, g, the random choices of $a, b \in \mathbb{Z}_p$, and $\mathcal{A}$'s coin tosses.

Oracle Diffie-Hellman (ODH) Assumption. In one concrete instantiation of our first PAPKE construction we use a variant of the Diffie-Hellman assumption called the *Oracle Diffie-Hellman (ODH)* assumption. Using the notation from above, a group $(\mathbb{G}, g, p)$ and a hash function $H : \mathbb{G} \to \{0,1\}^\kappa$ satisfy the ODH assumption if for all efficient adversaries $\mathcal{A}$, the probability

$$\big|\Pr[\mathcal{A}^{H_b(\cdot)}(\mathbb{G}, p, g, g^a, g^b, H(g^{ab})) = 1] - \Pr[\mathcal{A}^{H_b(\cdot)}(\mathbb{G}, p, g, g^a, g^b, r) = 1]\big|$$

is negligible in κ, where $H_b(\cdot)$ is defined as $H_b(x) = \begin{cases} H(x^b) & (x \neq g^a) \\ \perp & (x = g^a) \end{cases}$, and the probability is over the random choices of p, g, the random choices of $a, b \in \mathbb{Z}_p$ and $r \in \{0,1\}^\kappa$, and $\mathcal{A}$'s coin tosses.

5 Efficient & UC-Secure PAPKE Constructions

First attempts to construct PAPKE schemes that authenticate public keys and plaintexts with a password would probably involve message authentication codes

(MACs) of the public key and/or the enrypted plaintext under a key derived from the password. Such solutions, however, fall prey to offline dictionary attacks, either given just the authenticated public key, or by substituting the real public key with an adversarial one and testing the decrypted MAC. Thus the challenge is to devise schemes that withstand offline attacks and achieve the strong security guarantees formalized in our UC and property-based definitions. We present two very practical PAPKE constructions that achieve this goal.

The first construction, PAPKE-IC in Section 5.1, combines any CCA secure public-key encryption and an ideal cipher, using the ideal cipher to encrypt the public key with the password as a key. We prove this PAPKE scheme secure in the ideal-cipher model if the PKE scheme satisfies a number of properties that go beyond the standard CCA security, namely key anonymity, robustness, and the requirement that public keys are uniform in the (ideal) cipher domain.

While the PAPKE-IC construction is conceptually simple, instantiating the combination of ideal ciphers and public-key encryption requires some care, and subtle implementation mistakes could render the PAPKE-IC construction insecure (see the discussion in Section 5.1 below). Hence we propose a second PAPKE construction, PAPKE-FO in Section 5.2, which is not generic, but it does not need an ideal cipher and therefore might be easier to implement. It is based on a twin-key version of the Fujisaki-Okamoto transform of ElGamal encryption, and it is secure under the DDH assumption in ROM.

5.1 PAPKE-IC: Generic Construction from PKE and Ideal Cipher

Our first construction, protocol PAPKE-IC in Figure 4, builds PAPKE generically from a public-key encryption PKE and an ideal cipher $\mathsf{IC} = (\mathsf{IC.Enc}, \mathsf{IC.Dec})$. The basic idea of the construction is simple and similar to the Encrypted Key Exchange (EKE) PAKE of Bellovin and Merritt [9]: The receiver generates a key pair for the PKE scheme and encrypts the public key under the ideal cipher using the password as a key. The resulting encrypted public key is used as PAPKE authenticated public key *apk*. To encrypt a message, the sender decrypts *apk* under the ideal cipher using the password as a key, and encrypts the message under the resulting public key. Our PAPKE-IC shares this basic design with EKE, except that we use a CCA-secure encryption while EKE implicitely uses a version of (CPA-secure) ElGamal whose security *as encryption* is less clear.

Protocol PAPKE-IC requires a number of properties of the PKE scheme that go beyond the standard notion of indistinguishability under adaptive chosen-ciphertext attack. First, its public keys must be *uniformly distributed* over the domain of the ideal cipher, because otherwise an attacker can test passwords offline by trying to decrypt *apk*. Second, ciphertexts of the PKE cannot reveal under which public key they were encrypted, as that would allow offline attacks as well. The second property is known as key privacy or *anonymity* [6]. Third, and perhaps a bit harder to see, is that an adversary should be unable to construct ciphertexts that decrypt correctly under multiple secret keys, but such ciphertext would allow the adversary to test multiple password guesses in one query to the decryption oracle. This property is known as *strong robustness* [1]. The latter two

Setup: Let $\mathsf{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme with uniform public-key space $\mathcal{PK}$ and let $\mathsf{IC} = (\mathsf{IC.Enc}, \mathsf{IC.Dec})$ be an ideal cipher over $\mathcal{PK}$.

$\underline{\mathsf{PAPKE.KGen}(\kappa, pwd)}$:
Generate $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{PKE.KGen}(\kappa)$ and $apk \leftarrow \mathsf{IC.Enc}(pwd, pk)$ and output (sk, apk).

$\underline{\mathsf{PAPKE.Enc}(apk, pwd', m)}$:
Decrypt the public key, $pk \leftarrow \mathsf{IC.Dec}(pwd, apk)$ and output $c \leftarrow_{\mathrm{R}} \mathsf{PKE.Enc}(pk, m)$.

$\underline{\mathsf{PAPKE.Dec}(sk, c')}$:
Decrypt $m \leftarrow \mathsf{PKE.Dec}(sk, c')$ and output m.

Fig. 4. The generic PAPKE scheme PAPKE-IC.

properties are formalized in Section 4, as AI-CCA and SROB-CCA, respectively. Finally, PKE and IC have to be "compatible" in the sense that IC is an ideal cipher over the key space $\mathcal{PK}$ of PKE.

The proof of the following theorem is included in Appendix D:

Theorem 3. *Protocol* PAPKE-IC *in Figure 4 securely realizes functionality* $\mathcal{F}_{\mathsf{PAPKE}}$ *in the* $\mathcal{F}_{\mathsf{IC}}$*-hybrid model, if the public key encryption* PKE *has uniform public-key space* $\mathcal{PK}$ *and is* AI-CCA *and* SROB-CCA*-secure.*

Implementing Ideal Ciphers over Groups. Our PAPKE-IC construction assumes an ideal cipher over a key space $\mathcal{PK}$ that for many PKE schemes will be a cyclic group $\mathbb{G}$. We stress that such an assumption is also used in several PAKE schemes, beginning from the Bellare et al. analysis [8] of the Encrypted Key Exchange (EKE) PAKE scheme of Bellovin and Merritt [9]. Ideal ciphers over variable domains can be implemented for a variety of domains, e.g. [11]. However, for many groups implementing an ideal cipher is somewhat cumbersome and can introduce possibilities for offline and/or timing attacks. Simply applying a block cipher to the public key doesn't work as not all strings of the same length are valid group elements, and an adversary could offline tests by decrypting the authenticated public key under a guessed password and testing if the decryption yields a valid group element. If $\mathcal{PK} = \mathbb{G}$ is any elliptic curve group, there are deterministic methods that map any string onto a group element [10] and hence offline and timing attacks are not a concern. The opposite direction can be implemented as in [10], but that encoding works only for subspace S of roughly $1/2$ of $\mathbb{G}$ elements. This slows down key generation, i.e. pair $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{PKE.KGen}$ has to be chosen s.t. $pk \in S$, but it does not lead to timing attacks on passwords. Still, these mappings complicate key generation and are non-trivial to implement, which motivates searching for alternative solutions that do not rely on ideal ciphers over arbitrary groups.

DHIES-based Instantiation. In Section 6.1, we specify an efficient concrete instantiation of PAPKE-IC, called PAPKE-IC-DHIES, which uses a variant of DHIES as the robust and anonymous PKE. Scheme PAPKE-IC-DHIES is as efficient as one could hope for in a DH-based cryptosystem, i.e. it uses 1 exponentiation

in key generation, 2 exponentiations in encryption, and 1 in decryption. The DHIES variant we use (DHIES^*) was shown to satisfy the required properties under the Oracle-Diffie-Hellman assumption (ODH), using a collision-resistant hash function and a secure authenticated encryption scheme [1]. The authenticated encryption (or rather the combination of symmetric encryption and a MAC) needs to satisfy some additional, non-standard properties, and the ODH assumption also has an impact on the choice of the hash function. We refer to Section 6.1 for a more detailed discussion. Thus, similar to the challenges that arise when securely instantiating the ideal cipher, implementing DHIES^* also requires some care in the implementation and choice of its underlying primitives. Overall, this reliance of PAPKE-IC and PAPKE-IC-DHIES on non-standard assumptions motivates our second PAPKE protocol (Sec. 5.2) that trades the generic construction for simple and standard assumptions.

5.2 PAPKE-FO: Concrete Construction from DDH and ROM

Our second PAPKE construction, protocol PAPKE-FO in Figure 5, does not require an ideal cipher over a group of PKE public keys, or other building blocks with non-standard assumptions,nd may thus be easier to implement. It is however slightly more costly, with 2 exponentiations for key generation, 2 multi-exponentiations (with two bases) for encryption, and 1 exponentiation and 1 (two base) multi-exponentiation for decryption. This construction is built using the Fujisaki-Okamoto (FO) transform [20] for ElGamal encryption but with a "twin" Diffie-Hellman key instead of a single key.

The high-level idea is to derive the authenticated public key apk by "blinding" the public key g^x of the ElGamal encryption scheme with the hash of the password as $apk \leftarrow g^x \cdot \mathsf{H}_0(pwd)$, where H_0 is a hash function onto $\mathbb{G}$, which can be implemented in deterministic way (to avoid timing attacks) using e.g. [10]. To encrypt message m under password pwd' and key apk, the encryptor "unblinds" the public key as $y \leftarrow apk \cdot \mathsf{H}_0(pwd')^{-1}$ and then encrypts m under y using FO-ElGamal, i.e. the Fujisaki-Okamoto transform applied to ElGamal which lifs its security from CPA to CCA, required to achieve the CCA-security and ciphertext authenticity properties of PAPKE.

None of the password-derived values apk or c allows an offline attack: Any "unblinding" of apk would yield a valid public key g^x for some x, and ElGamal ciphertexts are known to guarantee key anonymity [6], meaning that ciphertexts do not leak information about the public key used in encryption. (Note that the leakage of the unblinded public key $y = g^x$ used in encryption would allow an adversary who sees $apk = y \cdot \mathsf{H}_0(pwd)$ to mount an offline attack on pwd.) The scheme is correct because if $pwd' = pwd$ then the hash values cancel and encryption is done under the "original" public key $y = g^x$. However, if the passwords do not match then encryption is done under an effectively random public key $y \leftarrow_{\mathrm{R}} \mathbb{G}$. The latter gives us the desired security against active attacks: If an honest party is tricked into encryption under a malicious apk^* but uses a different password than was used to blind apk^*, then the ciphertext will be indistinguishable from random, even if $\mathcal{A}$ knows the secret key to apk^*.

Note, however, that unlike the ideal cipher encryption of apk under pwd used in PAPKE-IC, the method used here to blind key g^x and form the authenticated key apk is essentially a one-time pad over $\mathbb{G}$, and thus is not by itself a commitment to password pwd. Below we discuss how we modify the above sketch and in particular make this blinding password-committing.

For the sake of simplicity in Figure 5 we describe PAPKE-FO for a message space $\mathcal{M} = \{0,1\}^n$ where n is the output length of H_2, but this can be extended to general message space $\mathcal{M} \leftarrow \{0,1\}^*$ using a flexible-output length hash function for H_2 (which can easily be built in the random oracle model), or using the hashed value as a key in a symmetric-key encryption.

Setup: Let $\mathbb{G}$ be a group of prime order p such that $2^{\kappa-1} < p < 2^\kappa$, and let g_1, g_2 be two random generators in $\mathbb{G}$. We use three hash functions modeled as random oracles: $\mathsf{H}_0 : \{0,1\}^* \to \mathbb{G}$, $\mathsf{H}_1 : \mathbb{G}^3 \times \{0,1\}^n \to \mathbb{Z}_p^2$ and $\mathsf{H}_2 : \mathbb{G} \to \{0,1\}^n$.

$\underline{\mathsf{PAPKE.KGen}(\kappa, pwd)}$:
- Choose $x \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ and compute $y_1 \leftarrow g_1^x,\ y_2 \leftarrow g_2^x,\ Y_2 \leftarrow y_2 \cdot \mathsf{H}_0(pwd)$.
- Output (sk, apk) for $sk \leftarrow (x, y_1, y_2)$ and $apk \leftarrow (y_1, Y_2)$.

$\underline{\mathsf{PAPKE.Enc}(apk, pwd', m)}$:
- Abort if $|m| > n$.
- Parse $(y_1, Y_2) \leftarrow apk$, and "unblind" the second public key, $y_2 \leftarrow Y_2 \cdot \mathsf{H}_0(pwd)^{-1}$.
- Generate randomness via the RO: compute $(r_1, r_2) \leftarrow \mathsf{H}_1(R, y_1, y_2, m)$ for $R \leftarrow_{\mathrm{R}} \mathbb{G}$.
- Encrypt R under y_1 and y_2: $c_1 \leftarrow g_1^{r_1} g_2^{r_2},\ c_2 \leftarrow y_1^{r_1} y_2^{r_2} \cdot R,\ c_3 \leftarrow \mathsf{H}_2(R) \oplus m$.
- Output $c \leftarrow (c_1, c_2, c_3)$.

$\underline{\mathsf{PAPKE.Dec}(sk, c')}$:
- Parse $(x, y_1, y_2) \leftarrow sk$.
- Decrypt $c = (c_1, c_2, c_3)$: $R \leftarrow c_2 / c_1^x$, $m \leftarrow c_3 \oplus \mathsf{H}_2(R)$, and $(r_1, r_2) \leftarrow \mathsf{H}_1(R, y_1, y_2, m)$.
- Verify the correctness of decryption: if $c_1 = g_1^{r_1} g_2^{r_2}$ set $m' \leftarrow m$, else set $m' \leftarrow \perp$.
- Output m'

Fig. 5. Our DDH-based PAPKE scheme PAPKE-FO.

Achieving UC-Security via "Twin" Keys. To achieve UC security we have to ensure that both the key apk and the ciphertext c commit each party to a well-defined password choice. Technically, the simulator SIM must be able to extract (i) pwd from an adversarial apk^* and (ii) pwd' and m from an adversarial ciphertext c. While (ii) can be realized via the Fujisaki-Okamoto transform, case (i) requires more care. We need (i) for the reasons outlined above, i.e. a ciphertext encrypted by an honest party under an adversarial key apk^* must be decryptable only if apk^* commits to the encryptor's password. In the UC functionality $\mathcal{F}_{\mathsf{PAPKE}}$ this is enforced by SIM having to pass a single password guess pwd^* corresponding to the real-life adversary's choice of apk^*, and if $pwd^* \neq pwd'$, i.e., the guess does not match the encryptor's password pwd', then the encryption must reveal no information on the encrypted plaintext.

We achieve this by generating a "twin" public key using two generators g_1, g_2 in the CRS. The *apk* then consists of $y_1 \leftarrow g_1^x$ and $Y_2 \leftarrow g_2^x \cdot \mathsf{H}_0(pwd)$, i.e., we keep one public key in the clear and the other one is blinded with the password hash. In the security proof we set $g_2 \leftarrow g_1^s$, which allows the simulator to decrypt $\mathsf{H}_0(pwd)$ from *apk*, and look up *pwd* from the random oracle queries. Further, encryption is done under *both* public keys: y_1 and the "unblinded" $y_2 = Y_2 \cdot \mathsf{H}_0(pwd)^{-1}$. This double encryption under the plain and derived key is crucial, as it prevents an adversary $\mathcal{A}$ from providing a malformed apk^* which would allow $\mathcal{A}$ to still decrypt, but from which SIM cannot extract a password. Thus, our "twin" key construction enforces that only a well-formed *apk* can lead to decryptable ciphertexts (if the passwords match), without requiring heavy tools such as zero-knowledge proofs.

The proof of the following theorem is included in Appendix E. In the security proof we assume that a common reference string functionality $\mathcal{F}_{\mathsf{CRS}}$ [16] provides all parties with the system parameters, and we model hash functions $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$ via a random oracle functionality $\mathcal{F}_{\mathsf{RO}}$ [24].

Theorem 4. *Protocol* PAPKE-FO *in Figure 5 securely realizes functionality* $\mathcal{F}_{\mathsf{PAPKE}}$ *under the DDH assumption in group* $\mathbb{G}$ *in the* $\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{RO}}$*-hybrid model.*

6 Concrete PAPKE and PAKE Instantiations

Here we show particular instantiations of some of our results, a PAPKE scheme called PAPKE-IC-DHIES, and two PAKE protocols, PAKE-IC-DHIES and PAKE-FO. PAPKE-IC-DHIES is a particular instantiation of the generic PAPKE-IC scheme of Section 5.1 based on the DHIES* PKE by Abdalla et el. [1], while the PAKE protocols are derived via the PAPKE-2-PAKE compiler of Section 3.1 applied to two concrete PAPKE schemes, PAPKE-IC-DHIES shown below and PAPKE-FO of Section 5.2.

6.1 Concrete Instantiation of PAPKE-IC Using DHIES

In Section 5.1 we show a generic UC-secure PAPKE scheme that relies on an ideal cipher and a public-key encryption scheme that is both AI-CCA and SROB-CCA-secure. Abdalla et al. [1] show that these properties can be realized by DHIES [2] modified such that it excludes zero randomness at encryption, i.e., samples r from $\mathbb{Z}_p^*$ instead of $\mathbb{Z}_p$, and rejects ciphertexts that have 1 as first component. This DHIES* scheme relies on authenticated encryption AE, a hash function H and a cyclic group $(\mathbb{G}, p, g)$ of prime order p and is defined as follows:

DHIES*.KGen(κ)**:** $x \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $y \leftarrow g^x$, set $pk \leftarrow y, sk \leftarrow x$ and return (pk, sk)

DHIES*.Enc(pk, m)**:** parse $pk = y$, get $r \leftarrow_{\mathrm{R}} \mathbb{Z}_p^*$, $k \leftarrow \mathsf{H}(y^r)$, $c_1 \leftarrow g^r$, $c_2 \leftarrow \mathsf{AE.Enc}(k, m)$ and return $c = (c_1, c_2)$.

DHIES*.Dec(sk, c)**:** parse $c = (c_1, c_2)$ and $sk = x$, get $k \leftarrow \mathsf{H}(c_1^x)$. If $c_1 = 1$ output $m \leftarrow \perp$ and $m \leftarrow \mathsf{AE.Dec}(k, c_2)$ else.

Lemma 1. *PKE* DHIES* *shown above is* AI-CCA *and* SROB-CCA*-secure if the Oracle-Diffie-Hellman assumption holds in* $\mathbb{G}$*,* H *is a collision-resistant hash, and* AE *is a secure, strongly unforgeable and collision-resistant authentication encryption scheme [1].*

There is a small caveat here: For historic reasons Abdalla et al. [1] have used a symmetric encryption scheme together with a MAC instead of an authenticated encryption we use in our exposition of DHIES*. Their result holds if MAC is collision-resistant and strongly unforgeable, i.e., we need the same properties to hold for the AE scheme as well. We refer to [1] for formal definitions of these additional assumptions, and to [2] for a detailed discussion of the Oracle-Diffie Hellman assumption and its impact on the choice of the hash function H. Scheme PAPKE-IC-DHIES shown in Figure 6 is a (semi) concrete instantiation of PAPKE-IC based on DHIES*. It requires only 2 exponentiations for encryption and 1 for decryption, and these costs dominate the costs of authenticated encryption as well as the ideal cipher over group $\mathbb{G}$ and hashing onto $\mathbb{G}$.

Setup: $\mathbb{G}$ is a cyclic group of prime order p with generator g; $\mathsf{IC} = (\mathsf{IC.Enc}, \mathsf{IC.Dec})$ is an Ideal Cipher over $\mathbb{G}$ with key space $\{0,1\}^*$, where $\mathsf{IC.Enc} : \{0,1\}^* \times \mathbb{G} \to \mathbb{G}$ and $\mathsf{IC.Dec} : \{0,1\}^* \times \mathbb{G} \to \mathbb{G}$; $\mathsf{AE} = (\mathsf{AE.Enc}, \mathsf{AE.Dec})$ is an authenticated encryption with key space $\{0,1\}^\kappa$; $\mathsf{H} : \mathbb{G} \to \{0,1\}^\kappa$ is a collision-resistant hash function.

$\underline{\mathsf{PAPKE.KGen}(\kappa, pwd)}$:

- Pick $x \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, compute $y \leftarrow g^x$ and $apk \leftarrow \mathsf{IC.Enc}(pwd, y)$.
- Assign $sk \leftarrow x$ and output (sk, apk).

$\underline{\mathsf{PAPKE.Enc}(apk, pwd', m)}$:

- Compute $y \leftarrow \mathsf{IC.Dec}(pwd, apk)$, $r \leftarrow_{\mathrm{R}} \mathbb{Z}_p^*$, $k \leftarrow \mathsf{H}(y^r)$, $c_1 \leftarrow g^r, c_2 \leftarrow \mathsf{AE.Enc}(k, m)$.
- Output $c = (c_1, c_2)$.

$\underline{\mathsf{PAPKE.Dec}(sk, c')}$:

- Parse $(c_1, c_2) \leftarrow c$, compute $k \leftarrow \mathsf{H}({c_1}^x)$.
- If $c_1 = 1$ set $m \leftarrow \perp$ otherwise set $m \leftarrow \mathsf{AE.Dec}(k, c_2)$, and output m.

Fig. 6. Concrete PAPKE instantiation PAPKE-IC-DHIES.

6.2 Concrete PAKE Protocols

We specify two concrete UC PAKE instantiations obtained by applying the generic PAPKE-2-PAKE compiler shown in Figure 2 to two PAPKE schemes: The first protocol, PAKE-IC-DHIES, uses PAPKE scheme PAPKE-IC-DHIES shown in Figure 6, and the second protocol, PAKE-FO, uses PAPKE scheme PAPKE-FO shown in Figure 5. Both PAKE protocols are UC-secure and highly efficient.

Indeed, to the best of our knowledge, these are the first two-round UC-secure PAKE's which rely on standard groups, i.e. no bilinear maps, but resort to the IC and/or ROM model to achieve practical efficiency. Concretely, PAKE-IC-DHIES uses 2 exponentiations per party and PAKE-FO uses 4 (multi-)exponentiations for one party and 2 for the other. This *almost* matches the efficiency and assumptions used by two-round PAKE's which were shown secure under only game-based security notions, e.g. [8, 13, 4], and it reduces from 3 to 2 the rounds of previously known UC PAKE secure under comparable assumptions of Abdalla et al. [3].

PAKE-IC-DHIES. Protocol PAKE-IC-DHIES shown in Figure 7 requires the same setup as the PAPKE scheme PAPKE-IC-DHIES in Figure 6, i.e. $\mathbb{G}$ is a cyclic group of prime order p with generator g (we assume $(\mathbb{G}, p, g)$ is available to all parties via a CRS functionality $\mathcal{F}_{\mathsf{CRS}}$), $\mathsf{IC} = (\mathsf{IC.Enc}, \mathsf{IC.Dec})$ is an ideal cipher over group $\mathbb{G}$ with key space $\{0,1\}^*$, where $\mathsf{IC.Enc} : \{0,1\}^* \times \mathbb{G} \to \mathbb{G}$ and $\mathsf{IC.Dec} : \{0,1\}^* \times \mathbb{G} \to \mathbb{G}$, $\mathsf{AE} = (\mathsf{AE.Enc}, \mathsf{AE.Dec})$ is an authenticated encryption with key space $\{0,1\}^\kappa$, and $\mathsf{H} : \mathbb{G} \to \{0,1\}^\kappa$ is a collision-resistant hash. The following security statement for PAKE-IC-DHIES follows from Theorem 2, Theorem 3, and Lemma 1:

Corollary 1. *The* PAKE-IC-DHIES *scheme described in Figure 7 securely realizes* $\mathcal{F}_{\mathsf{PAKE}}$ *in the* $\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{IC}}$*-hybrid model if the Oracle-Diffie-Hellman assumption is hard for* $\mathbb{G}$*,* H *is a collision-resistant hash, and* AE *is a secure, strongly unforgeable and collision-resistant authenticated encryption scheme.*

Party $\mathcal{P}_i$, upon input $(\mathsf{NEWSESSION}, sid', \mathcal{P}_i, \mathcal{P}_j, pwd, \text{client})$:	Party $\mathcal{P}_j$, upon input $(\mathsf{NEWSESSION}, sid', \mathcal{P}_j, \mathcal{P}_i, pwd', \text{server})$:
get $x \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $y \leftarrow g^x$	wait for message with prefix sid'
set $apk \leftarrow \mathsf{IC.Enc}(pwd, y)$	
store $sk \leftarrow x$	
$\xrightarrow{sid', apk}$	
	choose $k \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$
	get $y \leftarrow \mathsf{IC.Dec}(pwd', apk)$, $r \leftarrow_{\mathrm{R}} \mathbb{Z}_p^*$, $k' \leftarrow \mathsf{H}(y^r)$,
	$c_1 \leftarrow g^r, c_2 \leftarrow \mathsf{AE.Enc}(k', m)$, set $c \leftarrow (c_1, c_2)$
	output $(\mathsf{NEWKEY}, sid', k)$
$\xleftarrow{sid', c}$	
parse $c = (c_1, c_2)$, get $k' \leftarrow \mathsf{H}(c_1^x)$	
if $c_1 = 1$ set $m \leftarrow \perp$; else $m \leftarrow \mathsf{AE.Dec}(k', c_2)$	
if $m \neq \perp$, set $k \leftarrow m$; else $k \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$	
output $(\mathsf{NEWKEY}, sid', k)$	

Fig. 7. Two-round PAKE protocol PAKE-IC-DHIES.

PAKE-FO. Protocol PAKE-FO shown in Figure 8 requires the same setup as the PAPKE scheme PAPKE-FO in Figure 5, i.e. $\mathbb{G}$ is a group of prime order p

such that $2^{\kappa-1} < p < 2^{\kappa}$, g_1, g_2 are two random generators of $\mathbb{G}$ (we assume $(\mathbb{G}, p, g_1, g_2)$ is available to all parties via a CRS functionality $\mathcal{F}_{\mathsf{CRS}}$), and there are three hash functions $\mathsf{H}_0 : \{0,1\}^* \to \mathbb{G}$, $\mathsf{H}_1 : \mathbb{G}^3 \times \{0,1\}^n \to \mathbb{Z}_p^2$ and $\mathsf{H}_2 : \mathbb{G} \to \{0,1\}^n$, all implicitly prefixed with *sid*, which are implemented with a random oracle functionality $\mathcal{F}_{\mathsf{RO}}$. The following security statement for PAKE-FO follows from Theorem 2 and Theorem 4:

Corollary 2. *The* PAKE-FO *scheme described in Figure 8 securely realizes* $\mathcal{F}_{\mathsf{PAKE}}$ *in the* $\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{CRS}}$*-hybrid model if the DDH problem is hard in* $\mathbb{G}$*.*

Party $\mathcal{P}_i$, upon input (NEWSESSION, $sid', \mathcal{P}_i, \mathcal{P}_j, pwd$, client):	Party $\mathcal{P}_j$, upon input (NEWSESSION, $sid', \mathcal{P}_j, \mathcal{P}_i, pwd'$, server):
$x \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, $y_1 \leftarrow g_1^x$, $y_2 \leftarrow g_2^x$	wait for message with prefix sid'
$Y_2 \leftarrow y_2 \cdot \mathsf{H}_0(pwd)$, $apk \leftarrow (y_1, Y_2)$	
store $sk \leftarrow (x, y_1, y_2)$	
$\xrightarrow{sid', apk}$	
	choose $k \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$
	parse $apk = (y_1, Y_2)$, set $y_2 \leftarrow Y_2 \cdot \mathsf{H}_0(pwd')^{-1}$
	get $R \leftarrow_{\mathrm{R}} \mathbb{G}$, $(r_1, r_2) \leftarrow \mathsf{H}_1(R, y_1, y_2, k)$
	get $c_1 \leftarrow g_1^{r_1} g_2^{r_2}, c_2 \leftarrow y_1^{r_1} y_2^{r_2} \cdot R$,
	$c_3 \leftarrow \mathsf{H}_2(R) \oplus k$, set $c \leftarrow (c_1, c_2, c_3)$
	output (NEWKEY, sid', k)
$\xleftarrow{sid', c}$	
$R \leftarrow c_2 / c_1^x, k' \leftarrow c_3 \oplus \mathsf{H}_2(R)$,	
$(r_1, r_2) \leftarrow \mathsf{H}_1(R, y, y_1, y_2, k')$.	
if $c_1 = g_1^{r_1} g_2^{r_2}$, set $k \leftarrow k'$; else $k \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$	
output (NEWKEY, sid', k)	

Fig. 8. Two-round PAKE protocol PAKE-FO.

Acknowledgments. Anja Lehmann was supported by the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 786725 (OLYMPUS). Tatiana Bradley, Stanislaw Jarecki, and Jiayu Xu were supported by the NSF Cybersecurity Innovation for Cyberinfrastructure (CICI) Grant Award No. ACI-1547435.

References

1. Abdalla, M., Bellare, M., Neven, G.: Robust encryption. Cryptology ePrint Archive, Report 2008/440 (2008), `http://eprint.iacr.org/2008/440`
2. Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Heidelberg (Apr 2001)
3. Abdalla, M., Catalano, D., Chevalier, C., Pointcheval, D.: Efficient two-party passwordbased key exchange protocols in the uc framework. In: The Cryptographers Track at RSA Conference (CT-RSA) (2008)

4. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer, Heidelberg (Feb 2005)
5. Abu-Salma, R., Sasse, M.A., Bonneau, J., Danilova, A., Naiakshina, A., Smith, M.: Obstacles to the adoption of secure communication tools. In: 2017 IEEE Symposium on Security and Privacy. pp. 137–153. IEEE Computer Society Press (May 2017)
6. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 566–582. Springer, Heidelberg (Dec 2001)
7. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations among notions of security for public-key encryption schemes. In: Krawczyk, H. (ed.) CRYPTO'98. LNCS, vol. 1462, pp. 26–45. Springer, Heidelberg (Aug 1998)
8. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (May 2000)
9. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy. pp. 72–84. IEEE Computer Society Press (May 1992)
10. Bernstein, D.J., Hamburg, M., Krasnova, A., Lange, T.: Elligator: elliptic-curve points indistinguishable from uniform random strings. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 13. pp. 967–980. ACM Press (Nov 2013)
11. Black, J., Rogaway, P.: Ciphers with Arbitrary Finite Domains, pp. 114–130. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
12. Boneh, D., Gentry, C., Hamburg, M.: Space-efficient identity based encryption without pairings. In: 48th FOCS. pp. 647–657. IEEE Computer Society Press (Oct 2007)
13. Boyko, V., MacKenzie, P.D., Patel, S.: Provably secure password-authenticated key exchange using Diffie-Hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (May 2000)
14. Burr, W.E., Dodson, D.F., Newton, E.M., Perlner, R.A., Polk, W.T., Gupta, S., Nabbus, E.A.: Electronic authentication guideline. NIST Special Publication 800-63-1 (2011)
15. Camenisch, J., Lehmann, A., Neven, G., Samelin, K.: Uc-secure non-interactive public-key encryption. In: Computer Security Foundations Symposium, CSF 2017. pp. 217–233. IEEE Computer Society (2017)
16. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000), `http://eprint.iacr.org/2000/067`
17. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001)
18. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (May 2005)
19. Dai, Y., Steinberger, J.P.: Indifferentiability of 8-round Feistel networks. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 95–120. Springer, Heidelberg (Aug 2016)
20. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (Aug 1999)

21. Gentry, C., MacKenzie, P., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Advances in Cryptology – CRYPTO 2006. pp. 142–159. Springer (2006)
22. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences 28(2), 270–299 (1984)
23. Guardian: Whatsapp design feature means some encrypted messages could be read by third party. https://www.theguardian.com/technology/2017/jan/13/whatsapp-design-feature-encrypted-messages/ (2017)
24. Hofheinz, D., Müller-Quade, J.: Universally composable commitments using random oracles. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 58–76. Springer, Heidelberg (Feb 2004)
25. Holenstein, T., Künzler, R., Tessaro, S.: The equivalence of the random oracle model and the ideal cipher model, revisited. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd ACM STOC. pp. 89–98. ACM Press (Jun 2011)
26. Huima, A.: The Bubble Babble binary data encoding. http://web.mit.edu/kenta/www/one/bubblebabble/spec/jrtrjwzi/draft-huima-01.txt/ (2000)
27. Hwang, J.Y., Jarecki, S., Kwon, T., Lee, J., Shin, J.S., Xu, J.: Round-reduced modular construction of asymmetric password-authenticated key exchange. In: Security and Cryptography for Networks - SCN 2018. pp. 485–504 (2018)
28. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. Cryptology ePrint Archive, Report 2018/163 (2018), http://eprint.iacr.org/2018/163
29. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In: Advances in Cryptology - EUROCRYPT 2018 (2018)
30. Jutla, C., Roy, A.: Smooth NIZK arguments with applications to asymmetric UC-PAKE. Cryptology ePrint Archive, Report 2016/233 (2016), http://eprint.iacr.org/2016/233
31. Jutla, C.S., Roy, A.: Dual-system simulation-soundness with applications to UC-PAKE and more. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part I. LNCS, vol. 9452, pp. 630–655. Springer, Heidelberg (Nov / Dec 2015)
32. Katz, J., Vaikuntanathan, V.: Round-optimal password-based authenticated key exchange. Journal of Cryptology 26(4), 714–743 (Oct 2013)
33. Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 111–126. Springer, Heidelberg (Aug 2002)
34. OpenSSH 5.1 release announcement. https://www.openssh.com/txt/release-5.1/ (2008)
35. Rivest, R.L., Lampson, B.: SDSI – a simple distributed security infrastructure. http://people.csail.mit.edu/rivest/sdsi10.html/ (1996)
36. Tan, J., Bauer, L., Bonneau, J., Cranor, L.F., Thomas, J., Ur, B.: Can unicorns help users compare crypto key fingerprints? In: Mark, G., Fussell, S.R., Lampe, C., m. c. schraefel, Hourcade, J.P., Appert, C., Wigdor, D. (eds.) CHI Conference on Human Factors in Computing Systems. pp. 3787–3798. ACM (2017)
37. Tufekci, Z.: In response to guardians irresponsible reporting on whatsapp: A plea for responsible and contextualized reporting on user security. http://technosociology.org/?page_id=1687/ (2017)
38. Unger, N., Dechand, S., Bonneau, J., Fahl, S., Perl, H., Goldberg, I., Smith, M.: SoK: Secure messaging. In: 2015 IEEE Symposium on Security and Privacy. pp. 232–249. IEEE Computer Society Press (May 2015)

39. WhatsApp encryption overview: Technical white paper. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf/ (2016)

A Proof of Theorem 1 ($\mathcal{F}_{\mathsf{PAPKE}} \Rightarrow$ IND-CCKA + AUTH-CTXT)

We here give the proof of Theorem 1 of Section 2.2, stating that any PAPKE scheme $\Pi_{\mathsf{PAPKE}} = (\mathsf{KEYGEN}, \mathsf{ENCRYPT}, \mathsf{DECRYPT})$ that UC-realizes functionality $\mathcal{F}_{\mathsf{PAPKE}}$ is IND-CCKA secure and AUTH-CTXT secure. Notice that the syntax of Π_{PAPKE} and a PAPKE scheme $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ secure according to the IND-CCKA and AUTH-CTXT security definitions are different: the former algorithms take a session id as additional arguments in their inputs whereas the key generation and decryption algorithms of the latter output a secret key and take a secret key as input, respectively. Thus, we in the following consider IND-CCKA and AUTH-CTXT to be the definitions adapted to the algorithms $\Pi_{\mathsf{PAPKE}} = (\mathsf{KEYGEN}, \mathsf{ENCRYPT}, \mathsf{DECRYPT})$ with a dummy sid.

For both proofs we will replace the calls to algorithms $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ in the games of the property-based definitions to calls to the respective interfaces of $\mathcal{F}_{\mathsf{PAPKE}}|\mathsf{SIM}$, where SIM is the simulator specific to Π_{PAPKE}, and then analyse the success probability of the adversary. Viewing the property-based definition, as an environment interacting with either $\mathcal{F}_{\mathsf{PAPKE}}|\mathsf{SIM}$ or Π_{PAPKE}, this success probability in the modified and the unmodified games cannot differ by more than a negligible amount. The analysis of the adversary's success probability essentially boils down to the fact that the replies by $\mathcal{F}_{\mathsf{PAPKE}}$ to the calls made do not contain any information that could help the adversary to win the game – unless simulator a some point submits a password guess to $\mathcal{F}_{\mathsf{PAPKE}}$ that matches to password randomly chosen in the property-based definition. The probability that the simulator fails to do so is at least $(1 - q/|\mathcal{D}|)$, where q is the number of times $\mathcal{F}_{\mathsf{PAPKE}}$ allows the simulator to provide a password guess.

Proof of Theorem 1. We will first argue that any PAPKE scheme Π_{PAPKE} which UC-realizes functionality $\mathcal{F}_{\mathsf{PAPKE}}$ IND-CCKA secure and then show that it is also AUTH-CTXT secure.

IND-CCKA *Security.* Let SIM the simulator for $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ s.t. $\mathcal{F}_{\mathsf{PAPKE}}|\mathsf{SIM}$ and $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ are indistinguishable. For convenience, let us call the original IND-CCKA security game Game 0 and call Game 1 the one where $\mathcal{F}_{\mathsf{PAPKE}}|\mathsf{SIM}$ is called instead of the algorithms $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$. That is, In Game 1, instead of running $\mathsf{KGen}(\kappa, pwd)$, we call $\mathcal{F}_{\mathsf{PAPKE}}|\mathsf{SIM}$ on input $(\mathsf{KEYGEN}, sid, pwd)$ for some sid and the randomly chosen password pwd. We will get back an apk from $\mathcal{F}_{\mathsf{PAPKE}}|\mathsf{SIM}$. This call will succeed as $\mathcal{F}_{\mathsf{PAPKE}}|\mathsf{SIM}$ was not called before, so there cannot be any record $(\mathsf{badkeys}, sid, apk_j, \cdot)$ that could make this fail. When executing oracle LoR, instead of computing $C \leftarrow_{\mathrm{R}} \mathsf{Enc}(apk^*, pwd, m_b)$, we call $\mathcal{F}_{\mathsf{PAPKE}}|\mathsf{SIM}$ with $(\mathsf{ENCRYPT}, sid, apk^*, pwd, m_b^*)$ to obtain C. Finally, executing oracle Dec, instead of computing $m \leftarrow \mathsf{Dec}(sk, C)$ we call $\mathcal{F}_{\mathsf{PAPKE}}|\mathsf{SIM}$ with $(\mathsf{DECRYPT}, sid, C)$ to obtain m.

Let us further consider a Game 2 that is the same as Game 1 except that we change $\mathcal{F}_{\mathsf{PAPKE}}$ into $\mathcal{F}'_{\mathsf{PAPKE}}$ which considers all password guesses of SIM as being wrong (but is unchanged otherwise).

Let us next analyse the success probability of $\mathcal{A}$ in Game 2. The main observation is that the ciphertexts $\mathcal{F}'_{\mathsf{PAPKE}}|\mathsf{SIM}$ computes will be independent of the messages m_b. In fact, all messages $\mathcal{F}'_{\mathsf{PAPKE}}$ sends SIM will be independent of b and pwd and hence all ciphertext produced by SIM are independent of b. Therefore the success probability of $\mathcal{A}$ will be at most $1/2$.

Consider Game 1. This game will behave exactly as Game 2 unless pwd chosen by the IND-CCKA-security challenger matches one of the password guesses of SIM in Game 2. This happens with probability $(q_{apk^*} + q_{\mathsf{Dec}})/|\mathcal{D}|$ and thus the success probability of $\mathcal{A}$ in Game 1 will be

$$\begin{aligned}\Pr[\mathsf{Exp}^{\mathsf{Game1}}_{\mathcal{A},\mathsf{PAPKE}}(\kappa) = 1] &= \frac{1}{2}(1 - \frac{q_{apk^*} + q_{\mathsf{Dec}}}{|\mathcal{D}|}) + \delta(\frac{q_{apk^*} + q_{\mathsf{Dec}}}{|\mathcal{D}|}) \\ &\leq \frac{1}{2} + \frac{1}{2} \cdot \frac{q_{apk^*} + q_{\mathsf{Dec}}}{|\mathcal{D}|} \ ,\end{aligned}$$

where $0 \leq \delta \leq 1$ is the probability that $\mathcal{A}$ wins in case SIM sends a correct password guess to $\mathcal{F}_{\mathsf{PAPKE}}$. Because $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ securely realizes $\mathcal{F}_{\mathsf{PAPKE}}$, the differences in the success probabilities between Game 1 and Game 0 must be negligible (after all, $\mathcal{A}$ and the IND-CCKA game define an environment) and it follows that Π_{PAPKE} is IND-CCKA-secure.

AUTH-CTXT *Security.* Similarly as in the proof for IND-CCKA security, we construct Game 0, Game 1, and Game 2.

Let us now analyse the success probability of $\mathcal{A}$ in the Game 2. The winning condition is $\mathsf{Dec}(sk, C^*) \neq \perp$ and $C^* \notin \mathbf{L}$. Investigating the functionality, we can see that $\mathsf{Dec}(sk, C^*) = \perp$ if C^* has not been produced by the functionality and in the processing of the decryption call, the success probability will be 0.

Again, Game 1 behaves as Game 2 unless pwd chosen by the IND-CCKA-security challenger matches one of the password guesses of SIM in Game 2. Here, there are $q_{apk^*} + q_{\mathsf{Dec}} + 1$ occasions for SIM to provide a password. Thus the success probability of $\mathcal{A}$ in Game 1 is at most $\delta\frac{q_{apk^*}+q_{\mathsf{Dec}}+1}{|\mathcal{D}|}$, where $0 \leq \delta \leq 1$ is the probability that $\mathcal{A}$ wins in case SIM sends a correct password guess to $\mathcal{F}_{\mathsf{PAPKE}}$. From this it follows that the success probability in Game 0 will be $\Pr[\mathsf{Exp}^{\mathsf{AUTH\text{-}CTXT}}_{\mathcal{A},\mathsf{PAPKE}}(\kappa) = 1] \leq \frac{q_{apk^*}+q_{\mathsf{Dec}}+1}{|\mathcal{D}|} + \mathsf{negl}(\kappa)$.

□

B Ideal Functionalities for PKE and PAKE

We include two standard UC functionalities for reference. In Figure 9 we show functionality $\mathcal{F}_{\mathsf{PKE}}$ which is a standard UC model of PKE, and which forms the basis of our UC model of PAPKE defined via functionality $\mathcal{F}_{\mathsf{PAPKE}}$ given in Figure 1 in Section 2.2. In Figure 10 we show functionality $\mathcal{F}_{\mathsf{PAKE}}$ which

1. `Key Generation.` On input (KEYGEN, sid) from party $\mathcal{P}$:
   - If $sid \neq (\mathcal{P}, sid')$ or a record $(\mathsf{keyrec}, sid, \cdot)$ exists, ignore.
   - Send (KEYGEN, sid) to $\mathcal{A}$ and wait for $(\mathsf{KEYCONF}, sid, pk, \mathcal{M})$ from $\mathcal{A}$.
   - Create record $(\mathsf{keyrec}, sid, pk)$ and output $(\mathsf{KEYCONF}, sid, pk, \mathcal{M})$ to $\mathcal{P}$.
2. `Encryption.` On input $(\mathsf{ENCRYPT}, sid, pk', m)$ from party $\mathcal{Q}$ where $m \in \mathcal{M}$:
   - Retrieve record $(\mathsf{keyrec}, sid, pk)$.
   - If $pk' = pk$ and $\mathcal{P}$ (from $sid = (\mathcal{P}, sid')$) is honest:
     - Send $(\mathsf{ENC\text{-}L}, sid, pk, |m|)$ to $\mathcal{A}$ and wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
     - Abort if a record $(\mathsf{encrec}, sid, \cdot, c)$ exists.
     - Create record $(\mathsf{encrec}, sid, m, c)$.
   - Otherwise (i.e., if $pk' \neq pk$ or $\mathcal{P}$ is corrupt) :
     - Send $(\mathsf{ENC\text{-}M}, sid, pk', m)$ to $\mathcal{A}$ and wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
   - Output $(\mathsf{CIPHERTEXT}, sid, c)$ to $\mathcal{Q}$.
3. `Decryption.` On input $(\mathsf{DECRYPT}, sid, c)$ from party $\mathcal{P}$:
   - If $sid \neq (\mathcal{P}, sid')$ or no record $(\mathsf{keyrec}, sid, pk)$ exists, ignore.
   - If a record $(\mathsf{encrec}, sid, m, c)$ for c exists:
     - Output $(\mathsf{PLAINTEXT}, sid, c, m)$ to $\mathcal{P}$.
   - Else (i.e., if no such record exists):
     - Send $(\mathsf{DECRYPT}, sid, c)$ to $\mathcal{A}$ and wait for $(\mathsf{PLAINTEXT}, sid, m)$ from $\mathcal{A}$.
     - Create record $(\mathsf{encrec}, sid, m, c)$.
     - Output $(\mathsf{PLAINTEXT}, sid, c, m)$ to $\mathcal{P}$.

Fig. 9. Standard public-key functionality $\mathcal{F}_{\mathsf{PKE}}$ with message space $\mathcal{M}$.

is a standard UC model of Password-Authenticated Key Agreement (PAKE) [18]. We include the UC PAKE model because in Section 3.1 we show a generic compiler which uses UC PAPKE to build a UC PAKE protocol.

C Proof of Theorem 2 (Security of PAPKE-2-PAKE)

In this section we provide the full proof of Theorem 2, showing that our generic PAPKE-2-PAKE construction, shown in Figure 2 in Section 3.1, securely realizes the UC PAKE functionality $\mathcal{F}_{\mathsf{PAKE}}$ shown in Figure 10 in Appendix B. Figure 11 gives a formal presentation of the PAPKE-2-PAKE protocol in the $\mathcal{F}_{\mathsf{PAPKE}}$-hybrid model.

We start our proof in the "real world," in which the environment $\mathcal{Z}$, adversary $\mathcal{A}$ and players $\mathcal{P}_i$ and $\mathcal{P}_j$ run the real-world protocol based on the ideal functionality $\mathcal{F}_{\mathsf{PAPKE}}$. In the ideal world, the simulator SIM interacts with $\mathcal{F}_{\mathsf{PAKE}}$ and mimics the role of the real-world parties $\mathcal{P}_i/\mathcal{P}_j$ and the functionality $\mathcal{F}_{\mathsf{PAPKE}}$ towards $\mathcal{A}$. These two interactions, the real and the ideal, are shown conceptually in Fig. 12, and the goal of the proof is to show that the two interactions are indistinguishable in the environment's view.

The Simulator. In the description of the simulation algorithm SIM below we use "$\mathcal{F}_{\mathsf{PAPKE}}$" to refer to the simulator's sub-procedure shown in Figure 13.

1. `New Session.` On input $(\mathsf{NEWSESSION}, sid', \mathcal{P}_i, \mathcal{P}_j, pwd, \text{role})$ from party $\mathcal{P}_i$:
 – If this is the first NEWSESSION query, or if this is the second NEWSESSION query and there is a record $(\mathcal{P}_j, \mathcal{P}_i, pwd', \cdot, \cdot)$ then record $(\mathcal{P}_i, \mathcal{P}_j, pwd, \mathsf{fresh}, \perp)$.
 – Send $(\mathsf{NEWSESSION}, sid', \mathcal{P}_i, \mathcal{P}_j, \text{role})$ to $\mathcal{A}$.
2. `Password Guess.` On input $(\mathsf{TESTPWD}, sid', \mathcal{P}_i, pwd^*)$ from adversary $\mathcal{A}$:
 – Retrieve the record of the $(\mathcal{P}_i, \mathcal{P}_j, pwd, \mathsf{fresh}, \perp)$ (abort if no such record exists).
 – If $pwd^* = pwd$, then update the record to $(\mathcal{P}_i, \mathcal{P}_j, pwd, \mathsf{compromised}, \perp)$ and send $(\mathsf{TESTPWD}, sid', \texttt{correct})$ to $\mathcal{A}$.
 – Else, update record to $(\mathcal{P}_i, \mathcal{P}_j, pwd, \mathsf{interrupted}, \perp)$ and send $(\mathsf{TESTPWD}, sid', \texttt{wrong})$ to $\mathcal{A}$.
3. `Session Key.` On input $(\mathsf{NEWKEY}, sid', \mathcal{P}_i, k^*)$ from adversary $\mathcal{A}$ where $|k^*| = \kappa$:
 – Retrieve record $(\mathcal{P}_i, \mathcal{P}_j, pwd, \mathsf{status}, \perp)$ for $\mathsf{status} \in \{\mathsf{fresh}, \mathsf{interrupted}, \mathsf{compromised}\}$, abort if no such record exist.
 – If $\mathsf{status} = \mathsf{compromised}$ or either $\mathcal{P}_i$ or $\mathcal{P}_j$ is corrupted, set $k \leftarrow k^*$.
 – If $\mathsf{status} = \mathsf{fresh}$ and a record $(\mathcal{P}_j, \mathcal{P}_i, pwd, \mathsf{completed}, k')$ exist, set $k \leftarrow k'$.
 – Else set $k \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$.
 – Update the record to $(\mathcal{P}_i, \mathcal{P}_j, pwd, \mathsf{completed}, k)$, output $(\mathsf{NEWKEY}, sid', k)$ to $\mathcal{P}_i$.

Fig. 10. Standard PAKE functionality $\mathcal{F}_{\mathsf{PAKE}}$ [18] for security parameter κ.

Party $\mathcal{P}_i$, upon input $(\mathsf{NEWSESSION}, sid', \mathcal{P}_i, \mathcal{P}_j, pwd, \text{client})$:	Party $\mathcal{P}_j$, upon input $(\mathsf{NEWSESSION}, sid', \mathcal{P}_j, \mathcal{P}_i, pwd', \text{server})$:
$sid \leftarrow (\mathcal{P}_i, sid')$	wait for message with prefix sid'
input $(\mathsf{KEYGEN}, sid, pwd)$ to $\mathcal{F}_{\mathsf{PAPKE}}$	
obtain $(\mathsf{KEYCONF}, sid, apk)$ from $\mathcal{F}_{\mathsf{PAPKE}}$	
	$\xrightarrow{sid', apk}$
	choose $k \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$
	set $sid \leftarrow (\mathcal{P}_i, sid')$
	input $(\mathsf{ENCRYPT}, sid, apk, pwd', k)$ to $\mathcal{F}_{\mathsf{PAPKE}}$
	obtain $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{F}_{\mathsf{PAPKE}}$
	output $(\mathsf{NEWKEY}, sid', k)$
	$\xleftarrow{sid', c}$
input $(\mathsf{DECRYPT}, sid, c)$ to $\mathcal{F}_{\mathsf{PAPKE}}$	
obtain $(\mathsf{PLAINTEXT}, sid, m)$ from $\mathcal{F}_{\mathsf{PAPKE}}$	
if $m \neq \perp$, set $k \leftarrow m$; else $k \leftarrow_{\mathrm{R}} \{0,1\}^\kappa$	
output $(\mathsf{NEWKEY}, sid', k)$	

Fig. 11. Two-round PAKE protocol PAPKE-2-PAKE in the $\mathcal{F}_{\mathsf{PAPKE}}$-hybrid world.

This sub-procedure is a modification of the ideal PAPKE functionality code, i.e. algorithm $\mathcal{F}_{\mathsf{PAPKE}}$ shown in Figure 1, and it shows the heart of the simulator algorithm SIM in a way which makes it easier for us to argue that $\mathcal{A}$'s view of interacting with $\mathcal{P}_i/\mathcal{P}_j$ and $\mathcal{F}_{\mathsf{PAPKE}}$ is indistinguishable (indeed identical) to $\mathcal{A}$'s view of interaction with SIM which runs this "$\mathcal{F}_{\mathsf{PAPKE}}$" algorithm inside.

$\mathcal{P}_i$ first round: Upon receiving $(\mathsf{NEWSESSION}, sid', \mathcal{P}_i, \mathcal{P}_j, \text{client})$ from $\mathcal{F}_{\mathsf{PAKE}}$, SIM sets $sid \leftarrow (\mathcal{P}_i, sid')$ and runs "$\mathcal{F}_{\mathsf{PAPKE}}$" on input "$(\mathsf{KEYGEN}, sid, \perp)$ from $\mathcal{P}_i$" to generate apk. SIM then sends a network message (sid', apk), routed by $\mathcal{A}$, from "$\mathcal{P}_i$" to "$\mathcal{P}_j$".

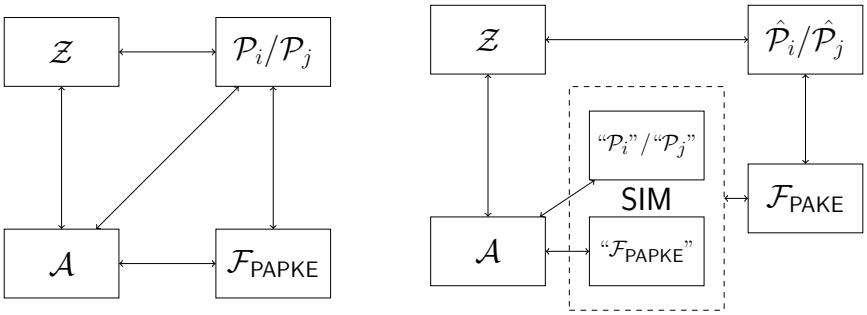


Fig. 12. Real world (left) vs. ideal world (right).

$\mathcal{P}_j$ second round and output: Upon receiving $(\mathsf{NEWSESSION}, sid', \mathcal{P}_i, \mathcal{P}_j, \text{server})$ from $\mathcal{F}_{\mathsf{PAKE}}$, SIM waits for a message (sid', apk') sent to "$\mathcal{P}_j$" from $\mathcal{A}$. (This is the message sent in the first round, with apk possibly modified by $\mathcal{A}$).

Then SIM chooses a session key $k \leftarrow_{\mathrm{R}} \{0,1\}^{\kappa}$, and runs "$\mathcal{F}_{\mathsf{PAPKE}}$" on input "$(\mathsf{ENCRYPT}, sid, apk', \perp, k)$ from $\mathcal{P}_j$" to obtain c.

SIM then sends $(\mathsf{NEWKEY}, sid', \mathcal{P}_j, k)$ to $\mathcal{F}_{\mathsf{PAKE}}$, as well as a network message (sid', c) from "$\mathcal{P}_j$" to "$\mathcal{P}_i$".

$\mathcal{P}_i$ final output: Upon receiving message (sid', c') sent to "$\mathcal{P}_i$" from $\mathcal{A}$, SIM sets $sid \leftarrow (\mathcal{P}_i, sid')$. It then runs "$\mathcal{F}_{\mathsf{PAPKE}}$" on input "$(\mathsf{DECRYPT}, sid, c')$ from $\mathcal{P}_i$", to obtain a plaintext m. SIM sets $k \leftarrow m$ and sends $(\mathsf{NEWKEY}, sid, \mathcal{P}_i, k)$ to $\mathcal{F}_{\mathsf{PAKE}}$.

$\mathsf{KEYGEN}, \mathsf{ENCRYPT}$, and $\mathsf{DECRYPT}$ queries to $\mathcal{F}_{\mathsf{PAPKE}}$ from corrupted parties:

If any valid query is made by a corrupted party to $\mathcal{F}_{\mathsf{PAPKE}}$, SIM runs "$\mathcal{F}_{\mathsf{PAPKE}}$" on the given inputs, and forwards any outputs back to that party. (Note that any query with $pwd = \perp$ is invalid, because $\perp \notin \mathcal{D}$).

The Proof. We show that the environment $\mathcal{Z}$'s view in the real world and its view in the ideal world are identical, i.e., that SIM's simulation is perfect. Figure 12 depicts the real-world and the ideal-world interactions. The left side of Fig. 12 shows the real world interaction, which is defined by the PAPKE-2-PAKE protocol executed by real honest parties $\mathcal{P}_i, \mathcal{P}_i$ and functionality $\mathcal{F}_{\mathsf{PAPKE}}$ which is used in that scheme. The right side of Fig. 12 shows the ideal-world interaction, which is defined by the $\mathcal{F}_{\mathsf{PAKE}}$ functionality (with dummy ideal-world players who merely pass messages to and from $\mathcal{F}_{\mathsf{PAKE}}$ and $\mathcal{Z}$) and the simulator SIM described above, who plays the role of the ideal-world adversary.

Below we analyze $\mathcal{Z}$'s view in these two security experiments, and we argue that the two worlds present identical views to $\mathcal{Z}$. Namely, for each message in $\mathcal{Z}$'s view we explain why in both worlds this message can be seen as created by the same interactive process. Without loss of generality, we assume that $\mathcal{A}$ is a "dummy" adversary that merely passes messages from and to $\mathcal{Z}$.

$\mathcal{P}_i$ first round:

1. `Key Generation.` On input $(\mathsf{KEYGEN}, sid, pwd)$ from party $\mathcal{P}$:
- If $sid \neq (\mathcal{P}, sid')$ or a record $(\mathsf{keyrec}, sid, \cdot, \cdot)$ exists, ignore.
- Send (KEYGEN, sid) to $\mathcal{A}$ and wait for $(\mathsf{KEYCONF}, sid, apk, \mathcal{M})$ from $\mathcal{A}$.
- If a record $(\mathsf{badkeys}, sid, apk_j, \cdot)$ with $apk_j = apk$ exists, abort.
- Create record $(\mathsf{keyrec}, sid, apk, pwd)$ and output $(\mathsf{KEYCONF}, sid, apk)$ to $\mathcal{P}$.

2. `Encryption.` On input $(\mathsf{ENCRYPT}, sid, apk', pwd', m)$ from party $\mathcal{Q}$ where $m \in \mathcal{M}$:
- If a record $(\mathsf{keyrec}, sid, apk, pwd)$ with $apk = apk'$ exists and $\mathcal{P}$ (from $sid = (\mathcal{P}, sid')$) is honest, then do the following:
  - Send $(\mathsf{ENC\text{-}L}, sid, |m|)$ to $\mathcal{A}$ and wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
  - Abort if a record $(\mathsf{encrec}, sid, \cdot, c)$ for c already exists.
  - Create record $(\mathsf{encrec}, sid, m', c)$ with $m' \leftarrow m$ if $pwd' = pwd$, and $m' \leftarrow \perp$ else.
- Otherwise do the following:
  - If a record $(\mathsf{badkeys}, sid, apk_j, pwd_j)$ with $apk_j = apk'$ exists, set $pwd^* \leftarrow pwd_j$.
  - Else, send $(\mathsf{GUESS}, sid, apk')$ to $\mathcal{A}$, wait for $(\mathsf{GUESS}, sid, pwd^*)$ from $\mathcal{A}$ and create record $(\mathsf{badkeys}, sid, apk', pwd^*)$.
  - Set $pwd^\bullet = pwd'$.
  - If $pwd' = \perp$, run $\mathsf{TESTPWD}(sid', \mathcal{P}, pwd^*)$ to obtain `answer`. If `answer` = `correct`, reset $pwd^\bullet \leftarrow pwd^*$. Otherwise, do nothing.
  - If $\boxed{pwd^\bullet} = pwd^*$, send $(\mathsf{ENC\text{-}M}, sid, m)$ to $\mathcal{A}$, wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
  - If $\boxed{pwd^\bullet} \neq pwd^*$, send $(\mathsf{ENC\text{-}L}, sid, |m|)$ to $\mathcal{A}$, wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
- If $pwd' \neq \perp$ and there exists a record $(\mathsf{keyrec}, sid, apk, \perp)$, create a record $(\mathsf{pwdguess}, sid, pwd', m, c)$.
- Output $(\mathsf{CIPHERTEXT}, sid, c)$ to $\mathcal{Q}$.

3. `Decryption.` On input $(\mathsf{DECRYPT}, sid, c)$ from party $\mathcal{P}$:
- If $sid \neq (\mathcal{P}, sid')$ or no record $(\mathsf{keyrec}, sid, apk, pwd)$ exists, ignore.
- If a record $(\mathsf{encrec}, sid, m, c)$ for c exists, where $m \in \mathcal{M} \cup \{\perp\}$:
  - Set $m' \leftarrow m$.
  - If a record $(\mathsf{pwdguess}, sid, pwd^*, m, c)$ exists, run $\mathsf{TESTPWD}(sid', \mathcal{P}, pwd^*)$. If `answer` = `wrong`, reset $m' \leftarrow \perp$. Otherwise, do nothing.
  - Output $(\mathsf{PLAINTEXT}, sid, \boxed{m'})$ to $\mathcal{P}$.
- Else (i.e., if no such record exists):
  - Send $(\mathsf{DECRYPT}, sid, c)$ to $\mathcal{A}$ and wait for $(\mathsf{PLAINTEXT}, sid, m, pwd^*)$ from $\mathcal{A}$.
  - Set $pwd^\bullet = pwd$.
  - If $pwd = \perp$, run $\mathsf{TESTPWD}(sid', \mathcal{P}, pwd^*)$. If `answer` = `correct`, reset $pwd^\bullet \leftarrow pwd^*$. Otherwise, do nothing.
  - If $pwd^* = \boxed{pwd^\bullet}$, set $m' \leftarrow m$ and $m' \leftarrow \perp$ otherwise.
  - Create record $(\mathsf{encrec}, sid, m', c)$.
  - Output $(\mathsf{PLAINTEXT}, sid, m')$ to $\mathcal{P}$.

Internal function $\mathsf{TESTPWD}(sid', \mathcal{P}, pwd)$:
- If record $(\mathsf{testrec}, sid', \mathcal{P}, \cdot)$ exists, set `answer` = `fail`.
- Otherwise, send $(\mathsf{TESTPWD}, sid', \mathcal{P}, pwd)$ to $\mathcal{F}_{\mathsf{PAKE}}$ to receive reply $(\mathsf{TESTPWD}, sid', \mathtt{answer})$ where `answer` $\in \{$`correct`, `wrong`$\}$. Create record $(\mathsf{testrec}, sid', \mathcal{P}, pwd, \mathtt{answer})$.
- Return `answer`.

Fig. 13. "$\mathcal{F}_{\mathsf{PAPKE}}$": Modified PAPKE functionality that is used as subprocedure of simulator SIM. Modifications from the original $\mathcal{F}_{\mathsf{PAPKE}}$ functionality are shown in boxes.

This round starts when $\mathcal{Z}$ sends $(\mathsf{NEWSESSION}, sid', \mathcal{P}_i, \mathcal{P}_j, pwd, \text{client})$ to $\mathcal{P}_i$. In both the real world and the ideal world, $\mathcal{Z}$'s query is ignored if a record $(\mathsf{keyrec}, sid, \cdot, \cdot)$ exists (i.e., $\mathcal{Z}$'s message is not the first NEWSESSION query for sid'). Otherwise $\mathcal{A}$ receives (KEYGEN, sid) from $\mathcal{F}_{\mathsf{PAPKE}}$ (or "$\mathcal{F}_{\mathsf{PAPKE}}$"). Then $\mathcal{A}$ replies with $(\mathsf{KEYCONF}, sid, apk)$. If a record $(\mathsf{badkeys}, sid, apk_j, \cdot)$ with $apk_j = apk$ exists, the session is aborted. Otherwise $\mathcal{A}$ receives (sid', apk) from $\mathcal{P}_i$ (or "$\mathcal{P}_i$"). $\mathcal{Z}/\mathcal{A}$'s views in the two worlds in this round are identical.

$\mathcal{P}_j$ second round and output:

This round starts when $\mathcal{Z}$ sends $(\mathsf{NEWSESSION}, sid', \mathcal{P}_i, \mathcal{P}_j, pwd', \text{server})$ to $\mathcal{P}_j$, and $\mathcal{A}$ sends (sid', apk') aimed at $\mathcal{P}_j$. (Note that the prefix of $\mathcal{A}$'s message must be sid', i.e., the same with the session id in $\mathcal{Z}$'s message; otherwise $\mathcal{P}_j$ will keep waiting and not proceed any further.)

In both worlds, the view of $\mathcal{Z}/\mathcal{A}$ in the second round depends on the inputs $\mathcal{A}$ provides.

If the network message (sid', apk') received by $\mathcal{P}_j$ has $apk' = apk$, where apk is the authenticated public key initially provided by $\mathcal{A}$ for that session, then $\mathcal{A}$ receives $(\mathsf{ENC\text{-}L}, sid, |m|)$ from $\mathcal{F}_{\mathsf{PAPKE}}$ (or "$\mathcal{F}_{\mathsf{PAPKE}}$"). Then $\mathcal{A}$ replies with $(\mathsf{CIPHERTEXT}, sid, c)$. If c is the result of a previous ENCRYPT query for apk, $\mathcal{F}_{\mathsf{PAPKE}}$ aborts and the session ends. Otherwise $\mathcal{P}_j$ (or "$\mathcal{P}_j$") sends (sid', c) to $\mathcal{P}_i$ (which is intercepted by $\mathcal{A}$), and outputs $(\mathsf{NEWKEY}, sid', k)$ where $k \leftarrow_{\mathrm{R}} \{0,1\}^{|m|}$. (In the real world, k is chosen by $\mathcal{P}_j$, while in the ideal world, it is chosen by $\mathcal{F}_{\mathsf{PAKE}}$.)

If $apk' \neq apk$, then we branch again:

(case I:) If a record $(\mathsf{badkeys}, sid, apk_j, pwd_j)$ such that $apk_j = apk'$ exists, then $\mathcal{A}$ receives $(\mathsf{ENC\text{-}M}, sid, k)$ from $\mathcal{F}_{\mathsf{PAPKE}}$ (or "$\mathcal{F}_{\mathsf{PAPKE}}$") if $pwd_j = pwd'$, and $(\mathsf{ENC\text{-}L}, sid, |m|)$ otherwise. (The process in the ideal world is as follows: SIM sends $(\mathsf{TESTPWD}, sid, \mathcal{P}_j, pwd_j)$ to $\mathcal{F}_{\mathsf{PAKE}}$. Since no TESTPWD query on $\mathcal{P}_j$ was sent to $\mathcal{F}_{\mathsf{PAKE}}$ previously, $\mathcal{P}_j$'s session record in $\mathcal{F}_{\mathsf{PAKE}}$ is fresh, so $\mathcal{F}_{\mathsf{PAKE}}$ replies with $(\mathsf{TESTPWD}, sid', \texttt{correct})$ if and only if $pwd_j = pwd'$. If $\mathcal{F}_{\mathsf{PAKE}}$'s response is $(\mathsf{TESTPWD}, sid', \texttt{correct})$, "$\mathcal{F}_{\mathsf{PAPKE}}$" sends $(\mathsf{ENC\text{-}M}, sid, k)$ to $\mathcal{A}$, otherwise it sends $(\mathsf{ENC\text{-}L}, sid, |m|)$ to $\mathcal{A}$.)

(case II:) Otherwise $\mathcal{A}$ receives $(\mathsf{GUESS}, sid, apk')$ from $\mathcal{F}_{\mathsf{PAPKE}}$ (or "$\mathcal{F}_{\mathsf{PAPKE}}$"), and replies with $(\mathsf{GUESS}, sid, pwd^*)$. If $pwd^* = pwd'$, $\mathcal{A}$ receives $(\mathsf{ENC\text{-}M}, sid, k)$ from $\mathcal{F}_{\mathsf{PAPKE}}$ (or "$\mathcal{F}_{\mathsf{PAPKE}}$"); otherwise it receives $(\mathsf{ENC\text{-}L}, sid, |m|)$. (The process in the ideal world is similar to the analysis in the case above.)

In both cases above, upon receiving $(\mathsf{ENC\text{-}M}, sid, k)$ or $(\mathsf{ENC\text{-}L}, sid, |m|)$, $\mathcal{A}$ replies with $(\mathsf{CIPHERTEXT}, sid, c)$. Then $\mathcal{P}_j$ (or "$\mathcal{P}_j$") sends (sid', c) to $\mathcal{P}_i$ (which is intercepted by $\mathcal{A}$), and outputs $(\mathsf{NEWKEY}, sid', k)$ where $k \leftarrow_{\mathrm{R}} \{0,1\}^{|m|}$.

$\mathcal{P}_i$ final output:

This round starts when $\mathcal{A}$ sends (sid', c') aimed at $\mathcal{P}_i$.

Again, in both worlds, the view of $\mathcal{Z}/\mathcal{A}$ in this round depends on the inputs provided by $\mathcal{A}$.

If a record $(\mathsf{encrec}, sid, k'/\perp, c')$ exists, i.e., c' is the result of a previous $(\mathsf{ENCRYPT}, sid, apk, \cdot, \cdot)$ query, then $\mathcal{Z}$ receives $(\mathsf{NEWKEY}, sid', k)$ from $\mathcal{P}_i$ (or

"$\mathcal{P}_i$"). Note that such ENCRYPT query may be performed by $\mathcal{P}_j$ in the previous round, or by a corrupted $\mathcal{P}^*$ generated by $\mathcal{Z}$ (separate from the PAKE protocol execution). We split this case into three subcases regarding how k is defined:

- In the previous round, $\mathcal{Z}$ sets $\mathcal{P}_j$'s password $pwd' = pwd$, and $\mathcal{A}$ passes $\mathcal{P}_i$'s message (sid', apk) to $\mathcal{P}_j$; in this round, $\mathcal{A}$ passes (sid', c') from $\mathcal{P}_j$ to $\mathcal{P}_i$ honestly. In the real world, $\mathcal{P}_j$ queries $(\mathsf{ENCRYPT}, sid, apk, pwd, k')$, where k' is a random string chosen by $\mathcal{P}_j$ and is the session key of both $\mathcal{P}_i$ and $\mathcal{P}_j$, i.e., $k = k'$. In the ideal world, SIM sends $(\mathsf{NEWKEY}, sid', \mathcal{P}_i, \perp)$ to $\mathcal{F}_{\mathsf{PAKE}}$. Since $\mathcal{P}_i$ and $\mathcal{P}_j$'s passwords are the same, $\mathcal{F}_{\mathsf{PAKE}}$ will make $\mathcal{P}_i$ and $\mathcal{P}_j$'s session keys the same, that is, $k = k'$.
- Same with the case above, except that $\mathcal{P}_j$'s password $pwd' \neq pwd$. In the real world, $\mathcal{P}_j$ queries $(\mathsf{ENCRYPT}, sid, apk, pwd, k')$, which generates a record $(\mathsf{encrec}, sid, \perp, c')$ in $\mathcal{F}_{\mathsf{PAPKE}}$. Then upon receiving $(\mathsf{DECRYPT}, sid, c')$ from $\mathcal{P}_i$, $\mathcal{F}_{\mathsf{PAPKE}}$ returns $(\mathsf{PLAINTEXT}, sid, \perp)$, and $\mathcal{P}_i$ sets $k \leftarrow_{\mathrm{R}} \{0,1\}^{|m|}$. In the ideal world, the process is similar with that in the case above, but since $\mathcal{P}_i$ and $\mathcal{P}_j$'s passwords are different, k is a random string in $\{0,1\}^{|m|}$.
- $\mathcal{Z}$ generates a corrupted $\mathcal{P}^*$, and queries $(\mathsf{ENCRYPT}, sid, apk, pwd, k')$ for any $k' \in \{0,1\}^{|m|}$ it chooses. In the real world, k is set to k'. In the ideal world, SIM sends $(\mathsf{TESTPWD}, sid, \mathcal{P}_i, pwd)$ to $\mathcal{F}_{\mathsf{PAKE}}$, and then $\mathcal{P}_i$'s session in $\mathcal{F}_{\mathsf{PAKE}}$ is compromised. After that, SIM sends $(\mathsf{NEWKEY}, sid', k')$ to $\mathcal{F}_{\mathsf{PAKE}}$, and $\mathcal{P}_i$'s session key k is set to k'.

In sum, in both worlds, $k = k'$ in the first and third subcases above, and $k' \leftarrow_{\mathrm{R}} \{0,1\}^{|m|}$ in the second subcase.

If a record $(\mathsf{encrec}, sid, k', c')$ does not exist, then $\mathcal{A}$ receives $(\mathsf{DECRYPT}, sid, c')$ from $\mathcal{F}_{\mathsf{PAPKE}}$ (or "$\mathcal{F}_{\mathsf{PAPKE}}$"), and replies with $(\mathsf{PLAINTEXT}, sid, k^*, pwd^*)$. After this, $\mathcal{Z}$ receives $(\mathsf{NEWKEY}, sid'', k)$ from $\mathcal{P}_i$ (or "$\mathcal{P}_i$"), where $k = k^*$ if $pwd^* = pwd$, and $k \leftarrow_{\mathrm{R}} \{0,1\}^{|m|}$ otherwise. (The process in the ideal world is as follows: SIM sends $(\mathsf{TESTPWD}, sid, \mathcal{P}_i, pwd^*)$ to $\mathcal{F}_{\mathsf{PAKE}}$. Since no TESTPWD query on $\mathcal{P}_i$ was sent to $\mathcal{F}_{\mathsf{PAKE}}$ previously, $\mathcal{P}_i$'s session record in $\mathcal{F}_{\mathsf{PAKE}}$ is fresh. If $pwd^* = pwd$, the record then becomes compromised, and $\mathcal{P}_i$'s session key is set to k^*. Otherwise the record becomes interrupted, and $\mathcal{P}_i$'s session key is a random string.)

D Proof of Theorem 3 (Security of PAPKE-IC)

We present the proof of Theorem 3, showing that the scheme PAPKE-IC described in Figure 4 securely realized our $\mathcal{F}_{\mathsf{PAPKE}}$ functionality in the ideal-cipher model if the underlying PKE scheme is indistinguishable, anonymous, and strongly robust (as defined in Section 4). To make the UC security proof more readable we re-present protocol PAPKE-IC in Figure 14 using the $\mathcal{F}_{\mathsf{PAPKE}}$ syntax and using functionality $\mathcal{F}_{\mathsf{PKE}}$ rather than a concrete PKE scheme as a tool. In the security proof we assume that the ideal cipher IC is made available to all parties through an ideal functionality $\mathcal{F}_{\mathsf{IC}}$, but for the ease of exposition use the algorithm-based

Setup: Let $\mathsf{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme with uniform public-key space $\mathcal{PK}$ and let $\mathsf{IC} = (\mathsf{IC.Enc}, \mathsf{IC.Dec})$ be an ideal cipher over $\mathcal{PK}$. We assume IC to be available to all parties through an ideal functionality $\mathcal{F}_{\mathsf{IC}}$.

$\underline{\mathsf{KEYGEN}}(sid, pwd)$:

- Abort if a key record $(sid, \cdot)$ already exists.
- Generate $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{PKE.KGen}(\kappa)$ and compute $apk \leftarrow \mathsf{IC.Enc}(pwd, pk)$.
- Store (sid, sk) and output $(\mathsf{KEYCONF}, sid, apk)$.

$\underline{\mathsf{ENCRYPT}}(sid, apk, pwd, m)$:

- Decrypt the public key: $pk \leftarrow \mathsf{IC.Dec}(pwd, apk)$.
- Encrypt the message $c \leftarrow_{\mathrm{R}} \mathsf{PKE.Enc}(pk, m)$ and output $(\mathsf{CIPHERTEXT}, sid, c)$.

$\underline{\mathsf{DECRYPT}}(sk, sid, c)$:

- Retrieve (sid, sk); abort if no such record exists.
- Decrypt $m \leftarrow \mathsf{PKE.Dec}(sk, c)$ and output $(\mathsf{PLAINTEXT}, sid, m)$.

Fig. 14. Scheme PAPKE-IC of Figure 4 represented in the $\mathcal{F}_{\mathsf{PKE}}$-hybrid model.

notation of IC instead of calling the sub-functionality with dedicated session identifiers.

The proof of Theorem 3 requires showing that for any environment $\mathcal{Z}$ and adversary $\mathcal{A}$ there exists an efficient ideal-world adversary algorithm, called simulator SIM, such that the view of $\mathcal{Z}$ of the real-world interaction, interacting with the real scheme and adversary $\mathcal{A}$, is indistinguishable from the view of $\mathcal{Z}$ functionality $\mathcal{F}_{\mathsf{PAPKE}}$ and simulator SIM.

Thus we first describe the behavior of our simulator SIM, and then we prove that the interaction with $\mathcal{A}$ and the real scheme is indistinguishable from an interaction with $\mathcal{F}_{\mathsf{PAPKE}}$ and SIM.

Simulator SIM. Simulator SIM keeps an initially empty list $\mathbf{L}$ and reacts to queries from the adversary or the functionality $\mathcal{F}_{\mathsf{PAPKE}}$ as follows:

Ideal Cipher: SIM simulates the ideal cipher $\mathsf{IC} = (\mathsf{IC.Enc}, \mathsf{IC.Dec})$ by lazy sampling while ensuring that $\mathsf{IC.Enc}(pwd', \cdot)$ is a permutation, thereby keeping a set $\mathbf{L}$ of tuples of the form (pwd', pk', apk', sk') where $\mathsf{IC.Enc}(pwd', pk') = apk'$ and $\mathsf{IC.Dec}(pwd', apk') = pk'$, respectively.

More precisely, upon a query $\mathsf{IC.Enc}(pwd', pk')$, SIM looks up whether an entry (pwd', pk', apk', sk') in $\mathbf{L}$ exists. If it is found, it returns apk'. Otherwise, it chooses $apk' \leftarrow_{\mathrm{R}} \mathcal{PK} \setminus \{apk'' \ : \ \exists (pwd', pk'', apk'', sk'') \in \mathbf{L}\}$. If there exists another tuple $(pwd'', pk'', apk'', sk'') \in \mathbf{L}$ with $apk'' = apk'$, then SIM aborts. Otherwise, it adds $(pwd', pk', apk', \perp)$ to $\mathbf{L}$ and returns apk'.

Upon a decryption query $\mathsf{IC.Dec}(pwd', apk')$, the simulator looks up $(pwd', pk', apk', sk') \in \mathbf{L}$ and returns pk' if it is found. If not, it generates key pairs $(pk', sk') \leftarrow_{\mathrm{R}} \mathsf{PKE.KGen}(\kappa)$ until there does not exist a tuple $(pwd', pk', apk', sk'') \in \mathbf{L}$. It adds (pwd', pk', apk', sk') to $\mathbf{L}$ and returns pk'.

Key Generation: Upon receiving (KEYGEN, sid) from $\mathcal{F}_{\mathsf{PAPKE}}$, SIM chooses $apk \leftarrow_{\mathrm{R}} \mathcal{PK}$, and sends $(\mathsf{KEYCONF}, sid, apk)$ to $\mathcal{F}_{\mathsf{PAPKE}}$.

Encryption: Here the behavior of SIM depends on whether the encryption is requested for the real apk or for a different $apk' \neq apk$. In the latter case, $\mathcal{F}_{\mathsf{PAPKE}}$ grants the simulator one password guess pwd^* for each apk'. If the guess matches the honest party's password attempt pwd', then the adversary learns the message to be encrypted, and only the length $|m|$ of the message otherwise.

- Real apk: If the encryption was triggered for the real apk, i.e., for the authenticated key that was sent by SIM to $\mathcal{F}_{\mathsf{PAPKE}}$ during key generation, then SIM receives $(\mathsf{ENC\text{-}L}, sid, apk, |m|)$ from $\mathcal{F}_{\mathsf{PAPKE}}$. SIM then chooses a dummy message m' such that $|m'| = |m|$, chooses $pk' \leftarrow_{\mathrm{R}} \mathcal{PK}$, and returns $c \leftarrow_{\mathrm{R}} \mathsf{PKE.Enc}(pk', m')$ to $\mathcal{F}_{\mathsf{PAPKE}}$ via the input $(\mathsf{CIPHERTEXT}, sid, c)$.
- Bad $apk' \neq apk$: If encryption is triggered for a different $apk' \neq apk$, then SIM receives $(\mathsf{GUESS}, sid, apk')$. It looks up a tuple $(pwd^*, pk', apk', \perp) \in \mathbf{L}$; because of how SIM simulates the ideal cipher, at most one such tuple exists. If no such tuple is found, it sets $pwd^* \leftarrow \perp$. The simulator then sends $(\mathsf{GUESS}, sid, pwd^*)$ to $\mathcal{F}_{\mathsf{PAPKE}}$.
  - If $pwd^* = pwd'$, i.e., SIM correctly guessed the password attempt pwd' of $\mathcal{Q}$, it receives $(\mathsf{ENC\text{-}M}, sid, apk', m)$ from $\mathcal{F}_{\mathsf{PAPKE}}$. SIM then computes an honest ciphertext $c \leftarrow_{\mathrm{R}} \mathsf{PKE.Enc}(pk', m)$ and returns it via $(\mathsf{CIPHERTEXT}, sid, c)$ to $\mathcal{F}_{\mathsf{PAPKE}}$.
  - If $pwd^* \neq pwd'$, i.e., SIM provided a wrong password guess, then it will only receive the length of the message from $\mathcal{F}_{\mathsf{PAPKE}}$ as $(\mathsf{ENC\text{-}L}, sid, apk', |m|)$. As in the case of encryption under an honest public key, SIM then creates a dummy ciphertext by choosing a dummy message m' such that $|m'| = |m|$, choosing $pk' \leftarrow_{\mathrm{R}} \mathcal{PK}$, and computing $c \leftarrow_{\mathrm{R}} \mathsf{PKE.Enc}(pk', 0^{|m|})$. It sends $(\mathsf{CIPHERTEXT}, sid, c)$ to the functionality.

Decryption: When $\mathcal{F}_{\mathsf{PAPKE}}$ is invoked by the honest party $\mathcal{P}$ to decrypt a ciphertext c that was not generated by the functionality, it asks the simulator for a plaintext m^* and a password guess pwd^* for the setup password pwd. Only if his guess is correct will the plaintext be returned, otherwise $\mathcal{F}_{\mathsf{PAPKE}}$'s output to $\mathcal{P}$ is $m = \perp$.

To determine the message m^* and password guess pwd^*, SIM first looks up whether a tuple $(pwd^*, pk', apk, sk') \in \mathbf{L}$ exists, such that $m = \mathsf{Dec}(sk', c) \neq \perp$. If multiple such tuples are found, the simulator aborts. If no such entry is found, it sets $m^* \leftarrow \perp$. Otherwise, it sets $m^* \leftarrow m$. Finally, the simulator sends $(\mathsf{PLAINTEXT}, sid, m^*, pwd^*)$ to $\mathcal{F}_{\mathsf{PAPKE}}$.

We now complete the proof by showing that the above simulator provides a correct simulation of the PAPKE-IC scheme through a sequence of games.

Proof. The proof goes by the standard technique of game changes which start from the real interaction between the environment $\mathcal{Z}$, adversary $\mathcal{A}$, and scheme

PAPKE-IC, and end with the ideal-world interaction between the same environment but interacting via functionality $\mathcal{F}_{\mathsf{PAPKE}}$ with the ideal-world adversary whose code is specified as the simulator SIM in Section 5.1.

Let $\mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{PAPKE\text{-}IC}}(\kappa)$ be the event that the UC environment $\mathcal{Z}$ outputs 1 in the $\mathcal{F}_{\mathsf{RO}}$-hybrid model with adversary $\mathcal{A}$ and protocol PAPKE-IC, and let $\mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathcal{F}_{\mathsf{PAPKE}}}(\kappa)$ be the event that $\mathcal{Z}$ outputs 1 when interacting in the ideal world with simulator SIM and the ideal functionality $\mathcal{F}_{\mathsf{PAPKE}}$. Then we will show that

$$|\Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{PAPKE\text{-}IC}}(\kappa)\right] - \Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathcal{F}_{\mathsf{PAPKE}}}(\kappa)\right]|$$

is negligible. In the following sequence of games, let $\mathbf{Gi}(\kappa)$ denote the event that $\mathcal{Z}$ outputs 1 when interacting with the experiment of Game i.

Game 1 *(Real world)*: The challenger runs the standard PAPKE-IC scheme in the role of an honest receiver $\mathcal{P}$ and honest senders $\mathcal{Q}_i$. It simulates the ideal cipher $(\mathsf{IC.Enc}, \mathsf{IC.Dec})$ by lazy sampling while ensuring that $\mathsf{IC.Enc}(pwd', \cdot)$ is a permutation, thereby keeping a set $\mathbf{L}$ of tuples of the form (pwd', pk', apk') where $\mathsf{IC.Enc}(pwd', pk') = apk'$ and $\mathsf{IC.Dec}(pwd', apk') = pk'$, respectively. Without loss of generality, we assume that the environment never makes the same ideal-cipher queries twice, and that it always makes a query $\mathsf{IC.Dec}(pwd', apk')$ before making an encryption query $(\mathsf{ENCRYPT}, sid, apk', pwd', m)$. We have that

$$\Pr\left[\mathbf{G1}(\kappa)\right] \;=\; \Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{PAPKE\text{-}IC}}(\kappa)\right] .$$

Game 2 *(Ideal decryption to public keys with known secret keys.)*: The simulator now stores tuples of the form (pwd', pk', apk', sk') to indicate that $\mathsf{IC.Enc}(pwd', pk') = apk'$ and $\mathsf{IC.Dec}(pwd', apk') = pk'$, and where sk' is either $\perp$ or the secret key corresponding to pk'. Namely, upon query $\mathsf{IC.Enc}(pwd', pk')$, store $(pwd', pk', apk', \perp)$ in $\mathbf{L}$ for a random apk' (while keeping $\mathsf{IC.Enc}(pwd', \cdot)$ a permutation), and upon query $\mathsf{IC.Dec}(pwd', apk')$, store (pwd', pk', apk', sk') for a freshly generated $(sk', pk') \leftarrow_{\mathrm{R}} \mathsf{KGen}(1^\kappa)$ (again, respecting permutation). This is merely a conceptual change, so we have that

$$\Pr\left[\mathbf{G2}(\kappa)\right] \;=\; \Pr\left[\mathbf{G1}(\kappa)\right] .$$

Game 3 *(Abort on ideal-cipher collisions)*: This game aborts when a pk' or apk' chosen at random during the simulation of a IC.Dec or IC.Enc query, respectively, collides with a pk' or apk' that was chosen at random for a different password. More specifically, this game aborts whenever there either exist two tuples $(pwd_1, pk_1, apk', \perp), (pwd_2, pk_2, apk', \perp) \in \mathbf{L}$ with $(pwd_1, pk_1) \neq (pwd_2, pk_2)$ (i.e., two different encryption queries have the same result apk'), or whenever there exist two tuples $(pwd_1, pk', apk_1, sk'), (pwd_2, pk', apk_2, sk') \in \mathbf{L}$ with $sk' \neq \perp$ and $(pwd_1, apk_1) \neq (pwd_2, apk_2)$ (i.e., two different decryption queries have the same result pk'). When q_{IC} denotes the total number of queries to IC.Enc and IC.Dec, we have that

$$|\Pr\left[\mathbf{G3}(\kappa)\right] - \Pr\left[\mathbf{G2}(\kappa)\right]| \;\leq \frac{q_{\mathrm{IC}}^2}{2 \cdot |\mathcal{PK}|} \quad .$$

GAME 4 *(Decrypting known ciphertexts via internal records)*: From now on, the challenger creates a record $(\mathsf{encrec}, sid, m', c)$ whenever it is triggered to encrypt a message m under $\mathcal{P}$'s public key apk and for password attempt pwd'. The ciphertext in this record is the standard encryption $c \leftarrow_{\mathrm{R}} \mathsf{PAPKE\text{-}IC.Enc}(apk, pwd', m)$. When $pwd' \neq pwd$, i.e., encryption is done for a different password than the challenger has chosen in the creation of apk, it sets $m' \leftarrow \perp$, otherwise it sets $m' \leftarrow m$. For decryption of some ciphertext c, the challenger first checks if a record exists. If so, it retrieves m from there (possibly being $\perp$), and only decrypts using the $\mathsf{PAPKE\text{-}IC.Dec}$ algorithm if no record was found. If more than one record for c exists, then the challenger aborts. Also, if the decryption algorithm $\mathsf{PAPKE\text{-}IC.Dec}(sk, c)$ returns $m \neq \perp$ for a ciphertext c that was created under a wrong password attempt $pwd' \neq pwd$, the challenger aborts. We distinguish between three different events causing the challenger to abort:

- $\mathbf{bad}_1$: There are two records $(\mathsf{encrec}, sid, m'_1, c)$ and $(\mathsf{encrec}, sid, m'_2, c)$ with $m'_1 \neq \perp$ and $m'_2 \neq \perp$. This means that c was the output to two inputs $(\mathsf{ENCRYPT}, sid, apk, pwd, m_1)$ and $(\mathsf{ENCRYPT}, sid, apk, pwd, m_2)$ for the correct password pwd. This is impossible by the correctness of PKE, so we have that $\Pr[\mathbf{bad}_1] = 0$.
- $\mathbf{bad}_2$: There are two records $(\mathsf{encrec}, sid, m'_1, c)$ and $(\mathsf{encrec}, sid, m'_2, c)$ with $m'_1 \neq \perp$ and $m'_2 = \perp$. An environment that causes this event to happen can be used to build an adversary $\mathcal{B}$ against the strong robustness of PKE as follows. Adversary $\mathcal{B}$, on input public keys (pk_0, pk_1), runs the environment and adversary $\mathcal{A}$ following the code of GAME 4, but rather than generating a fresh key pair $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{PKE.KGen}(\kappa)$ and setting $apk \leftarrow \mathsf{IC.Enc}(pwd, pk)$ on input $(\mathsf{KEYGEN}, sid, pwd)$ from the environment, it returns $apk \leftarrow \mathsf{IC.Enc}(pwd, pk_0)$, and uses its $\mathsf{Dec}(0, \cdot)$ oracle to respond to $\mathsf{DECRYPT}$ inputs. Also, it guesses an index $i \leftarrow_{\mathrm{R}} \{1, \ldots, q_{\mathrm{IC}}\}$ and responds the i-th ideal-cipher query with pk_1 instead of a freshly generated public key. If $\mathbf{bad}_2$ happens then c was output both by $\mathsf{ENCRYPT}(sid, apk, pwd, m_1)$ and by $\mathsf{ENCRYPT}(sid, apk, pwd', m_2)$, where $pwd' \neq pwd$. With probability $1/q_{\mathrm{IC}}$, we have that $\mathsf{IC.Dec}(pwd', apk) = pk_1$. This would imply that c was generated twice, once as $\mathsf{Enc}(pk_0, m_1)$ and once as $\mathsf{Enc}(pk_1, m_2)$. By the correctness of PKE, $\mathcal{B}$ can therefore output c as a ciphertext that will decrypt correctly under both keys, breaking the $\mathsf{SROB\text{-}CCA}$ security of PKE with probability

$$\mathbf{Adv}^{\mathsf{SROB\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}}(\kappa) \;=\; \frac{1}{q_{\mathrm{IC}}} \cdot \Pr[\mathbf{bad}_2] \; .$$

- $\mathbf{bad}_3$: The decryption algorithm $\mathsf{PAPKE\text{-}IC.Dec}(sk, c)$ returns $m \neq \perp$ for a ciphertext c that was created as $c \leftarrow_{\mathrm{R}} \mathsf{PAPKE\text{-}IC.Enc}(apk, pwd', m)$ for $pwd' \neq pwd$. An environment causing $\mathbf{bad}_3$ to happen gives rise to the following $\mathsf{SROB\text{-}CCA}$ adversary $\mathcal{B}$. On input (pk_0, pk_1), $\mathcal{B}$ runs the code of GAME 4, but sets $apk \leftarrow \mathsf{IC.Enc}(pwd, pk_0)$ and uses its $\mathsf{Dec}(0, \cdot)$ oracle to respond to $\mathsf{DECRYPT}$ inputs. It also guesses an index $i \leftarrow_{\mathrm{R}} \{1, \ldots, q_{\mathrm{IC}}\}$ and answers the i-th ideal-cipher query with pk_1. When $\mathbf{bad}_3$ occurs, then a ciphertext generated as $c \leftarrow_{\mathrm{R}} \mathsf{PKE.Enc}(\mathsf{IC.Dec}(pwd', apk), m)$ decrypts correctly to $m' \neq \perp$

under $sk = sk_0$. With probability $1/q_{\mathrm{IC}}$ we have that $\mathsf{IC.Dec}(pwd', apk) = pk_1$, so that, by the correctness of PKE, c decrypts correctly under both sk_0 and sk_1, so $\mathcal{B}$ outputs c. Its probability in breaking the $\mathsf{SROB\text{-}CCA}$ security of PKE is

$$\mathbf{Adv}^{\mathsf{SROB\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}}(\kappa) \;=\; \frac{1}{q_{\mathrm{IC}}} \cdot \Pr\left[\mathbf{bad}_3\right] .$$

In summary, we have that

$$\begin{aligned} \left|\Pr\left[\mathbf{G4}(\kappa)\right] - \Pr\left[\mathbf{G3}(\kappa)\right]\right| \;&\leq\; \Pr\left[\mathbf{bad}_1\right] + \Pr\left[\mathbf{bad}_2\right] + \Pr\left[\mathbf{bad}_3\right] \\ &=\; 2q_{\mathrm{IC}} \cdot \mathbf{Adv}^{\mathsf{SROB\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}}(\kappa) . \end{aligned}$$

Game 5 *(Decrypting unknown ciphertexts via the ideal-cipher history)*: Instead of normally decrypting ciphertexts for which no $(\mathsf{encrec}, sid, m', c)$ record exists, we extract m and pwd^* from the ideal-cipher queries. If $pwd^* = pwd$, the challenger returns m, otherwise it returns $\perp$. This game extracts a password pwd^* from ciphertext c by looking for a tuple $(pwd^*, pk', apk, sk') \in \mathbf{L}$ such that $\mathsf{Dec}(sk', c) = m \neq \perp$. If more than one such tuple is found, then we say that $\mathbf{bad}_4$ occurred and the game aborts. If no such tuple is found, then set $pwd^* = \perp$ and $m = \perp$, whereby $\mathbf{bad}_5$ denotes the event that $\mathsf{Dec}(sk, c) \neq \perp$ even though no suitable tuple was found.
If $\mathbf{bad}_4$ happens, then there are two tuples $(pwd_0^*, pk_0, apk, sk_0)$ and $(pwd_1^*, pk_1, apk, sk_1) \in \mathbf{L}$ such that $pwd_0^* \neq pwd_1^*$ and c decrypts validly under both pk_0 and pk_1. Given an environment that triggers $\mathbf{bad}_4$, one can build a $\mathsf{SROB\text{-}CCA}$ adversary $\mathcal{B}$ against PKE as follows. Adversary $\mathcal{B}$, on input public keys (pk_0, pk_1) chooses two random indices $i_0, i_1 \leftarrow_{\mathrm{R}} \{0, \ldots, q_{\mathrm{IC}}\}$. It runs the code of Game 4, but replaces the response of the i_b-th ideal-cipher query with pk_b for $b \in \{0, 1\}$ if $i_b > 0$, or if $i_b = 0$ replaces the public key pk generated at the beginning of the game with pk_b. In the latter case, it responds to further decryption queries using its own decryption oracle $\mathsf{Dec}(b, \cdot)$. If a ciphertext c is queried that triggers $\mathbf{bad}_4$ to happen, then with probability $\frac{2}{q_{\mathrm{IC}}(q_{\mathrm{IC}}+1)}$ the two tuples in $\mathbf{L}$ are exactly those where pk_0 and pk_1 were programmed, so that $\mathcal{B}$ can output c. We have that

$$\mathbf{Adv}^{\mathsf{SROB\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}}(\kappa) \;\geq\; \frac{2}{q_{\mathrm{IC}}(q_{\mathrm{IC}}+1)} \cdot \Pr\left[\mathbf{bad}_4\right] .$$

Event $\mathbf{bad}_5$ cannot happen because there always exists a tuple $(pwd, pk, apk, sk) \in \mathbf{L}$ for the real password pwd and the real public key apk, where $sk \neq \perp$ is the secret key corresponding to pk. We have that

$$\begin{aligned} \left|\Pr\left[\mathbf{G5}(\kappa)\right] - \Pr\left[\mathbf{G4}(\kappa)\right]\right| \;&\leq\; \Pr\left[\mathbf{bad}_4\right] + \Pr\left[\mathbf{bad}_5\right] \\ &\leq\; \frac{q_{\mathrm{IC}}(q_{\mathrm{IC}}+1)}{2} \cdot \mathbf{Adv}^{\mathsf{SROB\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}}(\kappa) . \end{aligned}$$

Game 6 *(Dummy plaintexts for real apk)*: In this game, when the challenger is asked to generate a ciphertext for a message m and password pwd' under the real

authenticated public key apk, it chooses a dummy message m' such that $|m'| = |m|$, chooses a random public key $pk' \leftarrow_{\mathrm{R}} \mathcal{PK}$, and returns $c \leftarrow_{\mathrm{R}} \mathsf{Enc}(pk', m')$. One can prove that this game is indistinguishable from the previous one under the AI-CCA security of PKE through a hybrid argument over the number of encryption queries q_{E}. In the i-th hybrid, the simulator responds the first $i-1$ encryption queries for apk with encryptions of a dummy message m' under a random public key pk', and responds the i-th to q_{E}-th queries with encryptions of the real message m under the real public key $pk' = \mathsf{IC.Dec}(pwd', apk)$. The first hybrid is identical to Game 5, while the $(q_{\mathrm{E}}+1)$-st hybrid is identical to Game 6. Given an environment that can distinguish the i-th hybrid from the $(i+1)$-st hybrid, one can build an AI-CCA adversary $\mathcal{B}$ that, on input public keys (pk_0, pk_1), guesses an ideal-cipher query index $j \leftarrow_{\mathrm{R}} \{0, \ldots, q_{\mathrm{IC}}\}$. If $j = 0$, then $\mathcal{B}$ uses pk_0 instead of a freshly generated pk to produce apk during key generation, and uses its $\mathsf{Dec}(0, \cdot)$ oracle to answer decryption queries. If the i-th encryption query $(\mathsf{ENCRYPT}, sid, apk', pwd', m)$ is not for the real $apk' = apk$ and the real password $pwd' = pwd$, then $\mathcal{B}$ aborts, otherwise it chooses a dummy message m' such that $|m'| = |m|$, submits (m, m') to its LoR oracle, and returns the resulting ciphertext c. Note that if the hidden bit b of the AI-CCA game is zero, then c is an encryption of m under pk_0, otherwise it is an encryption of m' under a fresh public key pk'. If the environment indicates that it is running in the i-th hybrid, then $\mathcal{B}$ outputs 0, otherwise it outputs 1. For the case that $j \neq 0$, $\mathcal{B}$ programs pk_0 as the output of the j-th ideal-cipher query, hoping that the i-th encryption query $(\mathsf{ENCRYPT}, sid, apk', pwd', m)$ is for $apk' = apk$ and $pwd' \neq pwd$ such that $\mathsf{IC.Dec}(pwd', apk) = pk_0$. The success probability of $\mathcal{B}$ is $1/(q_{\mathrm{IC}}+1)$ times the probability that the environment distinguishes the $(i-1)$-st hybrid from the i-th hybrid, so that the overall probability that the environment distinguishes Game 6 from Game 5 is

$$|\Pr[\mathbf{G6}(\kappa)] - \Pr[\mathbf{G5}(\kappa)]| \leq q_{\mathrm{E}}(q_{\mathrm{IC}}+1) \cdot \mathbf{Adv}^{\mathsf{AI\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}}(\kappa) .$$

Game 7 *(Dummy plaintexts for bad $apk' \neq apk$)*: In this game, we also respond to encryption queries $(\mathsf{ENCRYPT}, sid, apk', pwd', m)$ under a bad authenticated public key $apk' \neq apk$ with an encryption of a dummy message under a random key. However, we cannot do so for all such queries, because for a bad apk', the environment may actually know the corresponding secret key, so that it can detect the change. By Game 3, we have that apk' was only returned once as an output of a query $\mathsf{IC.Dec}(pwd^*, pk^*)$, meaning that there exists only one tuple $(pwd^*, pk^*, apk', \perp) \in \mathbf{L}$. If $pwd^* \neq pwd'$, we respond with a ciphertext $c \leftarrow_{\mathrm{R}} \mathsf{Enc}(pk', m')$ for a dummy message m' such that $|m'| = |m|$ and a random key $pk' \leftarrow_{\mathrm{R}} \mathcal{PK}$, otherwise we respond honestly with an encryption of m under pk^*.

We show the indistinguishability from the previous game through another hybrid on the number of encryption queries q_{E}. In the i-th hybrid, the game returns encryptions of dummy messages under random keys for the first $i-1$ encryption queries with $apk' \neq apk$ and $pwd' \neq pwd^*$, and honest encryptions for the other ones. An environment that distinguishes between two subsequent hybrid games i

and $i+1$ gives rise to the AI-CCA adversary $\mathcal{B}$ that, on input (pk_0, pk_1), guesses a query index $i \leftarrow_{\mathrm{R}} \{1, \ldots, q_{\mathrm{IC}}\}$ and returns pk_0 as the response to the i-th ideal-cipher query. For the i-th encryption query, it returns the response of its LoR oracle on (m, m'), where m' is a dummy message so that $|m'| = |m|$. If the environment indicates that it's in the i-th hybrid, then $\mathcal{B}$ outputs 0, otherwise it outputs 1. The success probability of $\mathcal{B}$ is $1/q_{\mathrm{IC}}$ times the probability that the environment distinguishes the $(i-1)$-st hybrid from the i-th hybrid, so that the overall probability that the environment distinguishes Game 6 from Game 5 is

$$|\Pr\left[\mathbf{G7}(\kappa)\right] - \Pr\left[\mathbf{G6}(\kappa)\right]| \;\leq\; q_{\mathrm{E}} \cdot q_{\mathrm{IC}} \cdot \mathbf{Adv}^{\mathsf{AI\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}}(\kappa) \;.$$

Game 8 *(Dummy apk)*: Rather than generating the real apk by first generating a key pair (pk, sk) and setting $apk \leftarrow \mathsf{IC.Enc}(pwd, pk)$, this game sets apk to be a random element in $\mathcal{PK}$. Note that the simulation at this point is independent of the real password pwd, except for the test whether $pwd^* = pwd$ in Game 5. If the environment makes a query $\mathsf{IC.Dec}(pwd, apk)$, then the simulator will generate a key pair $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{KGen}(1^\kappa)$ and program $\mathsf{IC.Dec}(pwd, apk) = pk$ so that the change is purely conceptional. If the environment never makes that query, then this entry will never be defined, but in that case also the secret key sk corresponding to $\mathsf{IC.Dec}(pwd, apk)$ is never needed in the simulation, so the change is again purely conceptional. (Note that Game 5 simply returns $\perp$ for ciphertexts that do not encrypt under any secret key in $\mathbf{L}$.) We have that

$$\Pr\left[\mathbf{G8}(\kappa)\right] \;=\; \Pr\left[\mathbf{G7}(\kappa)\right] \;.$$

Game 9 *(Challenger interacts with $\mathcal{F}_{\mathsf{PAPKE}}$)*: In our final game we make the transition from letting the challenger run the code of Game 8 to letting him run the simulator given in Section 5.1 in interaction with the ideal functionality $\mathcal{F}_{\mathsf{PAPKE}}$. As one can verify, this is merely a re-organization of code, so that

$$\Pr\left[\mathbf{G9}(\kappa)\right] \;=\; \Pr\left[\mathbf{G8}(\kappa)\right] \;=\; \Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathcal{F}_{\mathsf{PAPKE}}}(\kappa)\right] \;.$$

Adding all the probability differences between the games above yields

$$\begin{aligned} &|\Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathcal{F}_{\mathsf{PAPKE}}}(\kappa)\right] - \Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{PAPKE\text{-}IC}}(\kappa)\right]| \\ &\leq\; 2q_{\mathrm{E}}(q_{\mathrm{IC}}+1) \cdot \mathbf{Adv}^{\mathsf{AI\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}}(\kappa) + 2q_{\mathrm{IC}}(q_{\mathrm{IC}}+1) \cdot \mathbf{Adv}^{\mathsf{SROB\text{-}CCA}}_{\mathsf{PKE},\mathcal{B}}(\kappa) + \frac{q_{\mathrm{IC}}^2}{2 \cdot |\mathcal{PK}|} \;. \end{aligned}$$

□

E Proof of Theorem 4 (Security of PAPKE-FO)

We prove Theorem 4, i.e. that the PAPKE-FO scheme in Figure 5 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{PAPKE}}$ under the DDH assumption in the random oracle

model. The proof of Theorem 4 requires construction of simulator SIM s.t. for any environment $\mathcal{Z}$ and adversary $\mathcal{A}$ the view of $\mathcal{Z}$ in the real world where honest parties run $\mathsf{PAPKE\text{-}FO}$ with $\mathcal{A}$, is indistinguishable from the view in the ideal world where honest parties and SIM interact via functionality $\mathcal{F}_{\mathsf{PAPKE}}$. Thus we first show the simulator SIM and then we prove that the above holds for any efficient algorithms $\mathcal{Z}$ and $\mathcal{A}$.

Simulator SIM. Since many parts of SIM are similar with the simulator for the $\mathsf{PAPKE\text{-}IC}$ scheme, we only highlight the differences. Please refer to Appendix D for more details and intuitive explanations.

Setup (CRS and Random Oracles): SIM generates $\mathbb{G}, p$ and g_1 at random, but sets $g_2 \leftarrow g_1^s$ for $s \leftarrow_{\mathrm{R}} \mathbb{Z}_p$. (This allows SIM to decrypt pwd from malicious apk^*.)

SIM simulates the random oracles $\mathsf{H_0}, \mathsf{H_1}$ and $\mathsf{H_2}$ using lazy sampling as usual, but aborts when a collision $\mathsf{H_0}(q) = \mathsf{H_0}(q')$ or $\mathsf{H_1}(q) = \mathsf{H_1}(q')$ occurs for $q \neq q'$. SIM also aborts if two entries $\mathsf{H_1}(q) = (r_1, r_2)$ and $\mathsf{H_1}(q') = (r_1', r_2')$ exist such that $g_1^{r_1} g_2^{r_2} = g_1^{r_1'} g_2^{r_2'}$ for $q \neq q'$. The random oracles allow SIM to decrypt malicious ciphertexts for an honest apk, i.e., ones that are not generated via the ideal functionality without knowing the corresponding sk.

Key Generation: Upon receiving (KEYGEN, sid) from $\mathcal{F}_{\mathsf{PAPKE}}$, SIM creates $apk \leftarrow (y_1, Y_2)$ where $y_1, Y_2 \leftarrow_{\mathrm{R}} \mathbb{G}$ and sends $(\mathsf{KEYCONF}, sid, apk)$ to $\mathcal{F}_{\mathsf{PAPKE}}$. Thus, SIM does not know the secret key corresponding to apk.

Encryption: The simulation depends on whether ciphertexts for an honest or corrupt public key are requested.

real apk**:** Upon receiving $(\mathsf{ENC\text{-}L}, sid, apk, \perp)$ from $\mathcal{F}_{\mathsf{PAPKE}}$, SIM creates a dummy ciphertext $c = (c_1, c_2, c_3) \leftarrow_{\mathrm{R}} \mathbb{G} \times \mathbb{G} \times \{0,1\}^n$ and sends $(\mathsf{CIPHERTEXT}, sid, c)$ to $\mathcal{F}_{\mathsf{PAPKE}}$.

bad $apk' \neq apk$**:** Upon receiving $(\mathsf{GUESS}, sid, apk')$ from $\mathcal{F}_{\mathsf{PAPKE}}$, SIM parses $apk' = (y_1', Y_2')$ and "extracts" pwd^* from apk' by decrypting $K \leftarrow Y_2'/{y_1'}^s$ and looking up whether a query pwd^* was made where $\mathsf{H_0}(pwd^*) = K$. If no such query was found, it sets $pwd^* \leftarrow \perp$. SIM then sends $(\mathsf{GUESS}, sid, pwd^*)$ to $\mathcal{F}_{\mathsf{PAPKE}}$.

- Upon receiving $(\mathsf{ENC\text{-}M}, sid, apk', m)$ from $\mathcal{F}_{\mathsf{PAPKE}}$ (in the case where $pwd^* = pwd'$), SIM computes an honest ciphertext $c \leftarrow_{\mathrm{R}} \mathsf{Enc}(apk', pwd^*, m)$ and sends $(\mathsf{CIPHERTEXT}, sid, c)$ to $\mathcal{F}_{\mathsf{PAPKE}}$.
- Upon receiving $(\mathsf{ENC\text{-}L}, sid, apk', \perp)$ from $\mathcal{F}_{\mathsf{PAPKE}}$ (in the case where $pwd^* \neq pwd'$), SIM creates a dummy ciphertext $c \leftarrow_{\mathrm{R}} \mathbb{G} \times \mathbb{G} \times \{0,1\}^n$ and sends $(\mathsf{CIPHERTEXT}, sid, c)$ to $\mathcal{F}_{\mathsf{PAPKE}}$.

Decryption: Upon receiving $(\mathsf{DECRYPT}, sid, c)$ from $\mathcal{F}_{\mathsf{PAPKE}}$, SIM looks for a query $\mathsf{H_1}(R, y_1, y_2, m^*) = (r_1, r_2)$ for some y_2, m such that $c_1 = g_1^{r_1} g_2^{r_2}$. By the way we simulate $\mathsf{H_1}$ queries, there is at most one such query. If it exists, then SIM finds a query $\mathsf{H_0}(pwd^*) = Y_2/y_2$, of which there is also at most one. If any of these entries does not exist, then it sets $pwd^* = m^* = \perp$. Finally, SIM sends

(PLAINTEXT, sid, m^*, pwd^*) to $\mathcal{F}_{\mathsf{PAPKE}}$.

Proof. As in the proof of Theorem 3 we describe a sequence of games starting from the real interaction between the environment $\mathcal{Z}$, adversary $\mathcal{A}$, and scheme PAPKE-IC and ending in the ideal world where the environment interacts with the functionality $\mathcal{F}_{\mathsf{PAPKE}}$ and the simulator SIM described in Section 5.2.

In each game, we describe a challenger that interacts with a real-world adversary. The challenger plays the role of all honest parties (receiver $\mathcal{P}$ and senders $\mathcal{Q}_i$), obtaining their inputs from and passing their outputs to $\mathcal{Z}$. We stepwise change parts of the real PAPKE-FO protocol, mainly replacing real keys or ciphertexts either by "dummy" values that do not depend on pwd and m a anymore but that are indistinguishable from the real ones, or we derive them based only on knowing whether or not the passwords pwd and pwd' match. In our final game we then make the transition to let the challenger run internally the ideal functionality $\mathcal{F}_{\mathsf{PAPKE}}$ and simulate all messages based merely on the information he can obtain from $\mathcal{F}_{\mathsf{PAPKE}}$.

Let $\mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{PAPKE\text{-}IC}}(\kappa)$ be the event that the UC environment $\mathcal{Z}$ outputs 1 in the $\mathcal{F}_{\mathsf{RO}}$-hybrid model with adversary $\mathcal{A}$ and protocol PAPKE-IC, and let $\mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathcal{F}_{\mathsf{PAPKE}}}(\kappa)$ be the event that $\mathcal{Z}$ outputs 1 when interacting in the ideal world with simulator SIM and the ideal functionality $\mathcal{F}_{\mathsf{PAPKE}}$. Then we will show that

$$|\Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{PAPKE\text{-}FO}}(\kappa)\right] - \Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathcal{F}_{\mathsf{PAPKE}}}(\kappa)\right]|$$

is negligible. In the following sequence of games, let $\mathbf{Gi}(\kappa)$ denote the event that $\mathcal{Z}$ outputs 1 when interacting with the experiment of Game i.

Game 1: The challenger runs the standard PAPKE-FO scheme in the role of an honest receiver $\mathcal{P}$ and honest senders $\mathcal{Q}_i$. It simulates the random oracles $\mathsf{H}_0, \mathsf{H}_1$ and H_2 by running the code of $\mathcal{F}_{\mathsf{RO}}$, thereby keeping tables of queries and responses. These tables allow the challenger to check whether a certain query was made or look up preimages from previous responses. We have that $\Pr\left[\mathbf{G1}(\kappa)\right] = \Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{PAPKE\text{-}FO}}(\kappa)\right]$.

Game 2 *(Setting the CRS)*: The challenger changes the CRS and sets $g_2 \leftarrow g_1^s$. Clearly, this change has no impact on the view of the environment. (It will later enable the challenger to decrypt a password hash from a bad apk^*.) We have that $\Pr\left[\mathbf{G2}(\kappa)\right] = \Pr\left[\mathbf{G1}(\kappa)\right]$.

Game 3 *(Rejecting random-oracle collisions)*: This game aborts whenever a collision on one of the random oracles H_0 or H_1 arises, i.e., whenever there exist random oracle queries $q \neq q'$ such that $\mathsf{H}_0(q) = \mathsf{H}_0(q')$ or $\mathsf{H}_1(q) = \mathsf{H}_1(q')$. (From the random oracle H_2 we only require the observability property, but no collision-resistance and thus we do not check for collisions on H_2.) Given that $p > 2^{\kappa-1}$, the difference with the previous game for an adversary that makes q_{H} queries is $|\Pr\left[\mathbf{G3}(\kappa)\right] - \Pr\left[\mathbf{G2}(\kappa)\right]| \leq \frac{q_{\mathrm{H}}^2}{2^\kappa}$.

Game 4 *(Rejecting double representations)*: When answering H_1 queries, this game aborts whenever two entries $\mathsf{H}_1(q) = (r_1, r_2)$ and $\mathsf{H}_1(q') = (r_1', r_2')$ exist

such that $g_1^{r_1} g_2^{r_2} = g_1^{r'_1} g_2^{r'_2}$. Since the response (r_1, r_2) is chosen at random at the moment of each hash query, the group element $g_1^{r_1} g_2^{r_2}$ is uniformly distributed over $\mathbb{G}$. The difference between the games is therefore bounded by the probability that a group element is chosen twice when q_{H} elements are chosen at random, i.e., $|\Pr[\mathbf{G4}(\kappa)] - \Pr[\mathbf{G3}(\kappa)]| \leq \frac{q_{\mathrm{H}}^2}{2^\kappa}$. Note that from now on, every ciphertext $c = (c_1 = g_1^{r_1} g_2^{r_2}, c_2, c_3)$ corresponds to a unique (r_1, r_2) pair, which in turn corresponds to a unique (R, y_1, y_2, m) tuple (since we excluded collisions on H_1 in Game 3).

Game 5 *(Decrypting known ciphertexts via internal records)*: From now on, the challenger creates a record $(\mathsf{encrec}, sid, m', c)$ whenever it is triggered to encrypt a message m under $\mathcal{P}$'s public key apk and for password attempt pwd'. The ciphertext c in this record is the standard PAPKE-FO encryption of m under apk and pwd'. When $pwd' \neq pwd$, i.e., encryption is done for a different password than the challenger has chosen in the creation of apk, it sets $m' \leftarrow \perp$ and $m' \leftarrow m$ otherwise. For decryption of some ciphertext c, the challenger first checks if a record exists. If so, it retrieves m from there (possibly being $\perp$), and only decrypts normally if no record was found. If more than one record for c exist, the challenger aborts.
This change is not noticeable to the environment as long as (i) we have at most one record per ciphertext c, and (ii) PAPKE-FO will always output $\perp$ when decrypting a ciphertext that was created under a wrong password attempt. Recall that by Game 4, there can be only one query $\mathsf{H}_1(R, y_1, y_2, m) = (r_1, r_2)$ such that $c_1 = g_1^{r_1} g_2^{r_2}$. Condition (i) can therefore only be violated if the same value R is chosen twice during honest encryptions, which happens with probability at most $q_{\mathrm{E}}^2/2^\kappa$. Violating condition (ii) would imply that y_2 corresponding to c_1 is both $Y_2/\mathsf{H}_0(pwd')$ (it is computed this way in ENCRYPT) and $Y_2/\mathsf{H}_0(pwd)$ (because DECRYPT does not output $\perp$) for $pwd' \neq pwd$, meaning that $\mathsf{H}_0(pwd) = \mathsf{H}_0(pwd')$, which cannot happen because we excluded such collisions in Game 3. We therefore have that $|\Pr[\mathbf{G5}(\kappa)] - \Pr[\mathbf{G4}(\kappa)]| \leq \frac{q_{\mathrm{E}}^2}{2^\kappa}$.

Game 6 *(Decrypting unknown ciphertexts via the random-oracle history)*: When the challenger (in the role of $\mathcal{P}$) receives a ciphertext $c = (c_1, c_2, c_3)$, it looks for values R, y_2, and m such that an entry $\mathsf{H}_1(R, y_1, y_2, m) = (r_1, r_2)$ exists, where y_1 is the value included in apk, such that $c_1 = g_1^{r_1} g_2^{r_2}$. By Game 4, at most one such query exists. If no such entry exists, it sets $pwd^* = m = \perp$, which with overwhelming probability is a correct response because the probability that a future H_1 query will return (r_1, r_2) such that $g_1^{r_1} g_2^{r_2} = c_1$ for any of the q_{D} decryption queries is at most $q_{\mathrm{D}}/2^{\kappa-1}$.
Letting $h = Y_2/y_2$ where Y_2 is the group element included in apk, it then checks whether a query $\mathsf{H}_0(pwd^*) = h$ was made. By Game 3, at most one such entry exists, and the probability that any future random-oracle response hits h is at most $q_{\mathrm{D}}/2^{\kappa-1}$. If it exists, the challenger further tests whether it holds that $c_1 = g_1^{r_1} g_2^{r_2}$ and $c_2 = y_1^{r_1} y_2^{r_2} \cdot R$ and $c_3 = \mathsf{H}_2(R) \oplus m$. If so, and if moreover $pwd^* = pwd$, then the challenger creates a record $(\mathsf{encrec}, sid, m, c)$, otherwise it creates a record $(\mathsf{encrec}, sid, \perp, c)$. Subsequent decryptions of c are performed

simply by looking up m from there. The difference with the previous game is $|\Pr[\mathbf{G6}(\kappa)] - \Pr[\mathbf{G5}(\kappa)]| \leq \frac{q_D}{2^{\kappa-1}} + \frac{q_D}{2^{\kappa-1}} = \frac{q_D}{2^{\kappa-2}}$.

GAME 7 *(Dummy ciphertexts for real apk)*: In this game, when the challenger is asked to generate a ciphertext for a message m and password pwd' under the real apk, it returns a random tuple $c \leftarrow_R \mathbb{G} \times \mathbb{G} \times \{0,1\}^n$. We show that any adversary distinguishing this game from the previous one can be used to solve the CDH problem in $\mathbb{G}$.
If $c = (c_1 = g_1^{r_1} g_2^{r_2}, c_2 = y_1^{r_1}(Y_2/\mathsf{H}_0(pwd'))^{r_2} \cdot R, c_3 = \mathsf{H}_2(R) \oplus m)$ is a ciphertext that the challenger in GAME 6 generated for the real public key $apk = (y_1, Y_2)$ and some password pwd', then let us refer to $\mathsf{H}_1(R, y_1, Y_2/\mathsf{H}_0(pwd'), m)$ and $\mathsf{H}_2(R)$ as the "crucial" hash queries for c. It is easy to see that if the adversary does not make a "crucial" hash query for c, then due to the random choice of $R \leftarrow_R \mathbb{G}$, the ciphertext c is uniformly distributed over $\mathbb{G} \times \mathbb{G} \times \{0,1\}^n$. Any adversary that does make a "crucial" hash query can be turned into a solver $\mathcal{B}$ for the CDH problem in $\mathbb{G}$:
The algorithm $\mathcal{B}$, on input (g, A, B), follows the code of the game challenger except the followings: It responds to each random-oracle query $\mathsf{H}_0(pwd')$ by choosing $s_{pwd'} \leftarrow_R \mathbb{Z}_p$ and returning $A^{s_{pwd'}}$. It sets $g_1 \leftarrow g$, $y_1 \leftarrow A$, $Y_2 \leftarrow A^s \cdot \mathsf{H}_0(pwd)$, $apk \leftarrow (y_1, Y_2)$, where $s = \log_{g_1} g_2$ as has been the case since GAME 2. Finally, it picks $i \leftarrow_R \{1, \ldots, q_E\}$ (a guess for the order of encryption in which $\mathcal{A}$ makes the "crucial" hash query on the randomness R) and generates the i-th ciphertext for a message m_i on the real apk for a password pwd'_i by choosing $s_{1,i}, s_{2,i} \leftarrow_R \mathbb{Z}_p$, and returning $(c_{1,i} = B^{s_{1,i}} B^{s \cdot s_{2,i}}, c_{2,i} \leftarrow_R \mathbb{G}, c_{3,i} \leftarrow_R \{0,1\}^n)$. Note that the challenger thereby implicitly uses randomness $r_{1,i} = bs_{1,i}$ and $r_{2,i} = bs_{2,i}$ for the first part of the ciphertext. A crucial hash query for the i-th ciphertext must therefore contain the value for R such that $c_{2,i} = R \cdot y_1^{r_{1,i}}(Y_2/\mathsf{H}_0(pwd'_i))^{r_{2,i}} = R \cdot A^{bs_{1,i}}(A^{s - s_{pwd'_i}})^{bs_{2,i}}$, so $\mathcal{B}$ picks $R \leftarrow_R \{R' | \mathsf{H}_1(R', \cdot, \cdot, \cdot)$ or $\mathsf{H}_2(R)$ is queried$\}$ (a guess for R used in the i-th encryption) and outputs $(c_{2,i}/R)^{1/(s_{1,i} + s_{2,i}(s - s_{pwd'_i}))}$ as the CDH solution. It is easy to see that $\mathcal{B}$ simulates $\mathcal{A}$'s view perfectly unless and until $\mathcal{A}$ makes the "crucial" hash query (and then $\mathcal{B}$ outputs the CDH solution immediately, so there is no need to simulate anymore), and $\mathcal{B}$'s output is correct if both of its guesses on i and R are correct. Therefore, it outputs the correct solution with probability at least $1/q_H q_E$. We therefore have that $|\Pr[\mathbf{G7}(\kappa)] - \Pr[\mathbf{G6}(\kappa)]| \leq q_H q_E \cdot \mathbf{Adv}^{\mathrm{cdh}}_{\mathbb{G},\mathcal{B}}(\kappa)$.

GAME 8 *(Dummy ciphertexts for bad $apk' \neq apk$)*: In this game, we also create dummy ciphertexts when a bad key $apk' \neq apk$ is used in encryption and the password pwd^* embedded in apk' does not match the password pwd' used by the encryptor. That is, the environment provided an $apk' \neq apk$, to which he potentially also knows the secret key. However, if apk' is constructed for a password pwd^* which does not match the password attempt pwd' of the honest encryptor, we argue that he cannot distinguish honestly computed ciphertexts from dummy ones.
First, the challenger has to determine the password pwd^* embedded in apk', which he does by decrypting the ElGamal ciphertext $apk' = (y_1, Y_2)$ as $K =$

Y_2/y_1^s. We now check whether $K = \mathsf{H_0}(pwd')$. If so, we encrypt the message m normally under pwd' and apk' (so there is no change in this case). However, if the passwords did not match, or K is not a random oracle output, then we generate c as a random value in $\mathbb{G} \times \mathbb{G} \times \{0,1\}^n$.
Suppose $K \neq \mathsf{H_0}(pwd')$. Let $\mathsf{H_0}(pwd') = g_1^u$ and $K = g_1^v$ for some $u, v \in \mathbb{Z}_p$. Recall that the first two parts of the ciphertext are formed as $c_1 = g_1^{r_1} g_2^{r_2}$ and $c_2 = y_1^{r_1}(Y_2/\mathsf{H_0}(pwd'))^{r_2} = g_1^{xr_1+(sx+v-u)r_2}$ for random r_1, r_2. The values of s, x, u, and v are uniquely determined by g_2, y_1, $\mathsf{H_0}(pwd)$, and K, respectively, so the system of equations $r_1 + s \cdot r_2 = \log_{g_1} c_1$ and $x \cdot r_1 + (sx+v-u) \cdot r_2 = \log_{g_1} c_2$ is a system of two equations in two unknowns r_1 and r_2. The equations are linearly independent because the determinant of the system is $(sx + v - u) - sx \neq 0$ if $v - u \neq 0$. For any value of $\log_{g_1} c_1$ and $\log_{g_1} c_2$, this system therefore has a unique solution (r_1, r_2), or in other words, for any value of c_1, c_2, there exists exactly one choice for r_1, r_2 such that the encryption of m under apk' yields c_1 and c_2. The value of R is therefore information-theoretically hidden from the adversary, so that $c_3 = \mathsf{H_2}(R) \oplus m$ is also uniformly distributed. We therefore have that $\Pr[\mathbf{G8}(\kappa)] = \Pr[\mathbf{G7}(\kappa)]$.

Game 9 *(Dummy apk)*: Whereas until now the key generation was performed honestly as $apk = (y_1 = g_1^x, Y_2 = g_2^x \cdot \mathsf{H_0}(pwd))$ for secret key $(x, y_1, y_2 = g_2^x)$ with $x \leftarrow_{\mathrm{R}} \mathbb{G}$, we now let the challenger replace y_1 and Y_2 with random group elements $y_1, Y_2 \leftarrow_{\mathrm{R}} \mathbb{G}$ and set $apk = (y_1, Y_2)$. Since the value of x is no longer used to answer decryption queries, this change does not have any effect of the rest of the game. It is easy to see that any environment $\mathcal{Z}$ that can tell the difference with the previous game can be turned into a DDH-solving algorithm $\mathcal{B}'$ by, on input DDH problem instance (A, B, C), running $\mathcal{Z}$ with a game where $g_2 \leftarrow A$, $y_1 \leftarrow B$, and $Y_2 \leftarrow C \cdot \mathsf{H_0}(pwd)$. We therefore have that $|\Pr[\mathbf{G9}(\kappa)] - \Pr[\mathbf{G8}(\kappa)]| \leq \mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},\mathcal{B}'}(\kappa)$.

Game 10 *(Challenger interacts with $\mathcal{F}_{\mathsf{PAPKE}}$)*: In our final game we let the game challenger run the simulator SIM described in Section 5.2 in interaction with the ideal functionality $\mathcal{F}_{\mathsf{PAPKE}}$. As one can verify, this is merely a re-organization of code, so that $\Pr[\mathbf{G10}(\kappa)] = \Pr[\mathbf{G9}(\kappa)] = \Pr[\mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathcal{F}_{\mathsf{PAPKE}}}(\kappa)]$.

Adding all the probability differences between the games above yields

$$|\Pr[\mathbf{Ideal}_{\mathcal{Z},\mathsf{SIM},\mathcal{F}_{\mathsf{PAPKE}}}(\kappa)] - \Pr[\mathbf{Real}_{\mathcal{Z},\mathcal{A},\mathsf{PAPKE\text{-}FO}}(\kappa)]| \leq q_{\mathrm{H}} q_{\mathrm{E}} \cdot \mathbf{Adv}^{\mathrm{cdh}}_{\mathbb{G},\mathcal{B}}(\kappa) + \mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},\mathcal{B}'}(\kappa) + \frac{2q_{\mathrm{H}}^2 + q_{\mathrm{E}}^2 + 4q_{\mathrm{D}}}{2^\kappa} .$$

As the hardness of the CDH problem is implied by the hardness of the DDH problem, Theorem 4 follows. □