Public-Key Encryption with Non-interactive Opening

Ivan Damgård[1], Dennis Hofheinz[2], Eike Kiltz[⋆2], and Rune Thorbek[1]

[1] BRICS, Aarhus
[2] CWI, Amsterdam

Abstract. We formally define the primitive of public-key encryption with non-interactive opening (PKENO), where the receiver of a ciphertext C can, convincingly and without interaction, reveal what the result was of decrypting C, without compromising the scheme's security. This has numerous applications in cryptographic protocol design, e.g., when the receiver wants to demonstrate that some information he was sent privately was not correctly formed. We give a definition based on the UC framework as well as an equivalent game-based definition. The PKENO concept was informally introduced by Damgård and Thorbek who suggested that it could be implemented based on Identity-Based Encryption. In this paper, we give direct and optimized implementations, that work without having to keep state information, unlike what one obtains from directly using IBE.

1 Introduction

Motivation. Consider the following extremely common scenario from cryptographic protocol design: Player A sends a secret message to player B who—perhaps at some later time—checks what he receives against some public information. For instance, it may be that the message is supposed to be information for opening a commitment that A established earlier. If the check is OK, B will be able to proceed, but otherwise some "exception handling" must be done. The standard solution to this is to have B broadcast a complaint, and A must then broadcast what he claims to have sent privately, allowing all players to check the information. This is secure, since the conflict can only occur if at least one of A,B is corrupt, so the adversary already knows what is broadcast. But it has the important drawback that interaction is required, in particular A must be present to help resolve the conflict. In many cases, one cannot rely on this assumption. For instance, suppose A is one of many clients who want to provide some input to a set of servers, who will then do a secure computation on the inputs. It is highly desirable that this can be done without interaction, in particular that the servers can decide efficiently among themselves which clients provided well-formed input.

Public-key encryption with non-interactive opening. An alternative solution was suggested by Damgård and Thorbek in [12], namely *public-key encryption with non-interactive opening* (PKENO). This is based on the observation that in practice, the private communication from A to B would typically be implemented using public key encryption, i.e., A sends a ciphertext C encrypted under B's public key pk_B. PKENO now means that if B chooses to reveal the result m of decrypting C (typically, if he is unhappy about m), he can do so, convincingly and without interaction[3]. That is, he can broadcast m, π where π is a proof that can be checked against pk_B and C and demonstrates that indeed decrypting C using the secret key matching pk_B results in m. Of course, this

⋆ Supported by the research program Sentinels (`http://www.sentinels.nl`). Sentinels is being financed by Technology Foundation STW, the Netherlands Organization for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs.

[3] Note that m may not be a meaningful message, it may be a special reject symbol if C was rejected as invalid by the decryption algorithm.

must be done such that other ciphertexts remain secure, and this excludes the trivial solution of revealing B's secret key. Clearly, if PKENO can be implemented efficiently, we have a nice general tool for removing interaction from cryptographic protocols.

Difficulty of PKENO. Note that having the receiver open a ciphertext is less trivial than having the sender do so: the sender could always be asked to simply reveal the plaintext and the random coins used to construct the ciphertext. This does not work when the receiver does the opening: one has to consider the fact that the sender might be corrupt and hence C is adversarially constructed. It may not even be a valid ciphertext, in which case "the coins used to construct C" is not a well defined concept.

Inefficient Constructions. A few straightforward solutions for implementing PKENO do exist which, however, have various drawbacks: In principle, one can implement PKENO if a common reference string can be reliably set up. Then the receiver B can commit to his secret key initially and π would be a non-interactive zero-knowledge proof that the secret key committed to matches pk_{B} and produces m when used to decrypt C. Unfortunately, with the known techniques for non-interactive zero-knowledge, this solution is very inefficient and essentially useless in practice. Efficient solutions are easy to find in the random oracle model, since one can take known efficient and interactive zero-knowledge proofs and make them non-interactive using the Fiat-Shamir heuristic. However, it is unclear what security guarantees in the random oracle model mean for the real world, so in this paper, we will concentrate on efficient solutions that do not use random oracles.

Known Constructions and their Limitations. In [12], the PKENO notion was informally introduced, and it was suggested that it could be implemented based on identity-based encryption (IBE). The idea here is that pk_{B} would be the public master key of an IBE system, and the secret key sk_{B} would be the secret master key. To encrypt m, one chooses an identity id (see below for details on how id is chosen), and encrypts m to this identity. Thus, the ciphertext C is the pair $C = (id, \mathsf{IBEenc}(id, m))$. The receiver B uses sk_{B} to generate the IBE user secret key $usk[id]$ corresponding to id and can then decrypt. To open C, B simply reveals the decryption result m and $usk[id]$, this allows anyone to do the decryption and check that the result is m. Note that efficient IBE schemes exist (under specific assumptions) that do not require random oracles [21, 14].

It follows directly from the properties of IBE that revealing $usk[id]$ does not compromise security of ciphertexts directed to other identities, not even if id is adversarially chosen. This solution is therefore secure if we can guarantee that identities cannot be reused — but only then. This would be the case if it is used in a protocol that assigns unique labels to all ciphertexts to be sent. Then these labels can be used as identities. But note that these labels must be different in different instances of the same protocol. Alternatively, all players could keep state information allowing to test if a label has been used before. This puts some rather heavy demands on the implementation and hence, using IBE in this straightforward way is not satisfactory in general.

An alternative construction of PKENO can be obtained by using *public-key encryption with witness-recovering decryption* (PKEWR) [20]. Here the receiver (i. e., the holder of the secret key) can efficiently reconstruct the "randomness" that was used for encryption. This randomness then serves as the proof. Verification performs (deterministic) re-encrypting using the randomness and the messages. The proof is valid if the result equals the ciphertext. There exists construction of PKEWR from the Decisional Diffie-Hellman assumption and from an assumption related to lattices. However, both constructions are relatively inefficient since the ciphertext size is linear in the message length.

Our Contributions. In this paper, we make two contributions: first, we give a formal definition of PKENO, in fact we give two equivalent definitions, one based on the UC framework, and a game-based definition. This allows to show that an implementation is secure using the game-based definition (which is usually easier than with UC), while at same time being guaranteed the composition properties that follows from the UC theorem. We assume — for simplicity — a trusted key set-up, i.e., all key pairs are correctly generated. We emphasize, however, that this assumption is not inherent to the PKENO concept. The definitions can be modified to do without it and some implementations do not need it.

Second, we show some concrete implementations of PKENO. One of our techniques gives a simple and general solution to the problem with unique identities in the IBE implementation, allowing a stateless solution. To this end we use a technique by Naor and Yung [18] that was also used more recently by Canetti, Halevi, and Katz [9] in a transformation of any chosen-plaintext secure IBE scheme into a chosen-ciphertext secure PKE scheme. We adopt the latter transformation to construct PKENO from IBE. The idea is to use, for each PKENO encryption, a fresh random verification key of a one-time signature scheme as the "identity" *id* for IBE encryption. In order to tie the IBE ciphertext to this verification key it is signed using the corresponding signing key. This ensures the uniqueness of the identity and hence allows a stateless solution of PKENO.

Another technique gives a more direct implementation that is not based on IBE and hence is more efficient. We use a modification of the pairing-based chosen-ciphertext secure PKE scheme which was proposed by Boyen, Mei, Waters [5] and Kiltz [16]. We show that it is possible to update their scheme with a non-interactive opening functionality without compromising its security. Security of this scheme can be reduced to the Bilinear Decisional Diffie-Hellman (BDDH) assumption.

2 Preliminaries

2.1 Notational conventions

If x is a string, then $|x|$ denotes its length, while if S is a set then $|S|$ denotes its size. If $k \in \mathbb{N}$ then 1^k denotes the string of k ones. If S is a set then $s \leftarrow_{\mathrm{R}} S$ denotes the operation of picking an element s of S uniformly at random. Unless otherwise indicated, algorithms are randomized and polynomial time. An adversary is an algorithm or a tuple of algorithms. A function $f : \mathbb{N} \to \mathbb{R}$ is *negligible* iff there exists $c < 0$ such that $|f(k)| < k^c$ for all sufficiently large k. We write $f \approx g$ if $f - g$ is negligible.

2.2 The UC model

Canetti's Universal Composability (UC) framework [6, 7] for multi-party computation allows to formulate security and composition of multi-party protocols in a very general way. The idea of the UC model is to compare a protocol to an idealization of the respective protocol task. Security means that the protocol "looks like" the idealization even in face of arbitrary attacks and in arbitrary protocol environments. This notion of security is very strict [8, 2, 13], but implies useful compositional properties [6]. In fact, in a certain sense, this notion is even necessary for secure composition of protocols [17].

The real model. We shortly outline the framework for multi-party protocols defined in [6, 7]. First of all, parties (denoted by P_1 through P_n) are modeled as interactive Turing machines (ITMs) (cf. [7]) and are supposed to run some fixed protocol (i.e., program) Π. There also is an adversary, denoted $\mathcal{A}$ and modeled as an ITM as well, that carries out attacks on protocol Π. Therefore, $\mathcal{A}$ may corrupt parties (in which case it learns the party's state and controls its future actions), and intercept or inject messages sent between parties. If $\mathcal{A}$ corrupts parties only before the actual protocol run of Π takes place, $\mathcal{A}$ is called non-adaptive, otherwise $\mathcal{A}$ is said to be adaptive. In this work, we only consider non-adaptive corruptions. The respective local inputs for all parties of protocol Π are supplied by an environment machine (modeled as an ITM and denoted $\mathcal{Z}$), which may also read all protocol outputs locally made by the parties and communicate with the adversary.

The ideal model. The model we have just described is called the real model of computation. In contrast to this, the ideal model of computation is defined just like the real model, with the following exceptions: all party ITMs are replaced with one single ideal functionality $\mathcal{F}$. The ideal functionality may not be corrupted by the adversary, yet may send messages to and receive messages from it. Finally, the adversary in the ideal model is called "simulator" and denoted $\mathcal{S}$. The only means of attack the simulator has in the ideal model are corruptions (in which case $\mathcal{S}$ may supply inputs to and read outputs from $\mathcal{F}$ in the name of the corrupted party), delaying or suppressing outputs of $\mathcal{F}$, and all actions that are explicitly specified in $\mathcal{F}$. However, $\mathcal{S}$ has no access to the inputs $\mathcal{F}$ gets and to the outputs $\mathcal{F}$ generates, nor are there any protocol messages to intercept. Intuitively, the ideal model of computation (or, more precisely, the ideal functionality $\mathcal{F}$ itself) should represent what one ideally expects the protocol to do. In fact, for a number of standard tasks, there are formulations as such ideal functionalities (see, e.g., [6]).

Security definition. To decide whether or not a given protocol Π fulfills the requirements of our ideal specification $\mathcal{F}$, the framework of [6] uses a simulatability-based approach: at a time of its choice, $\mathcal{Z}$ may halt and generate output. The random variable describing the first bit of $\mathcal{Z}$'s output will be denoted by $\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(k,z)$ when $\mathcal{Z}$ is run with security parameter $k \in \mathbb{N}$ and initial input $z \in \{0,1\}^*$ in the real model of computation, and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)$ when $\mathcal{Z}$ is run in the ideal model. Now Π is said to *securely realize* $\mathcal{F}$ iff for any real adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$ and any (possibly non-uniform) family of initial inputs $z = (z_k)_k$, we have

$$\Pr\left[\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(k,z_k) = 1\right] \approx \Pr\left[\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z_k) = 1\right]. \tag{1}$$

This slightly differs from the original formulations in [6, 7], but is equivalent and eases our presentation. Intuitively, Equation 1 means that any attack against the protocol can be simulated in the ideal model. Hence, any weakness of the real protocol is already contained in the ideal specification (that does not contain an "actual" weakness by definition). Interestingly, the "worst" real attack possible is the one carried out by the dummy adversary $\tilde{\mathcal{A}}$ that simply follows $\mathcal{Z}$'s instructions. That means that for security, it actually suffices to demand existence of a simulator that simulates attacks carried out by $\tilde{\mathcal{A}}$.

Composition of protocols. To formalize the composition of protocols, there also exists a model "in between" the real and ideal model of computation. Namely, the hybrid model of computation is identical to the real model, except that parties have access to (multiple instances of) an ideal functionality that aids in running the protocol. This is written as $\varphi^{\mathcal{F}}$ for the actual protocol φ and the ideal functionality $\mathcal{F}$. Instances of $\mathcal{F}$ are distinguished via session identifiers (short: session ids, or *sid*s). Note that syntactically, instances of $\mathcal{F}$ can be implemented by a protocol Π geared towards realizing $\mathcal{F}$. And indeed, the universal composition theorem [6, 7] guarantees that if one protocol instance of Π is secure, then many protocol instances are, even when used in arbitrary protocols φ. More concretely, if Π securely realizes $\mathcal{F}$, then φ^{Π} securely realizes $\varphi^{\mathcal{F}}$ for any protocol φ. Here, $\varphi^{\mathcal{F}}$

denotes that φ uses (up to polynomially many) instances of $\mathcal{F}$ as a subprimitive, and φ^{Π} denotes that φ uses instances of Π instead.

Conditional security and composability. Universal composability is a very strict notion. So sometimes (e.g., in the case of bit commitments), it is not possible to achieve full UC security. Hence, several weakenings of the notion have been proposed. One method that will be useful in our case is to consider only protocol environments that conform to certain rules (see [19, 1]). Concretely, secure realization with respect to a certain class $\mathfrak{Z}$ of environments means that in Equation 1, we quantify only over environments in $\mathfrak{Z}$. This relaxed security notion still gives precisely those compositional guarantees one would expect: secure composition with larger protocols that can be seen as restricted environments from $\mathfrak{Z}$ (see [19, 1] for details).

3 Public-key encryption with non-interactive opening

3.1 A UC-based definition

Figure 1 depicts our ideal functionality for public-key encryption with non-interactive openings. $\mathcal{F}_{\text{PKENO}}$ is an extension of the $\mathcal{F}_{\text{PKE}}$ functionality [6, 10, 15] that captures IND-CCA secure public-key encryption. The most notable difference to $\mathcal{F}_{\text{PKE}}$ are the additional `Prove` and `Verify` queries, which allow the receiver to open a ciphertext and every party to verify openings. Also, we dropped public keys, since we assume a trusted PKI (i.e., keypair setup) for a realization.

Discussion of $\mathcal{F}_{\text{PKENO}}$. First, note that the session id *sid* already determines the distinguished receiving party P_{recv}. Any party may ask for encryptions, but only P_{recv} may ask for decryptions. As for the encryption of a message m, the adversary may determine a unique tag C via the algorithm Enc. However, note that C depends only on the length $|m|$ of m, but not on m itself (except if the receiver is corrupted, in which case we obviously cannot guarantee secrecy). This reflects that ideally, encryptions reveal only the length of the message. Decryption takes care that correctness is ensured, i.e., ciphertexts are mapped back to the encrypted messages. (For this, $\mathcal{F}_{\text{PKENO}}$ stores a list of ciphertexts and associated messages.)

Opening and verifying openings is a bit trickier. For any ciphertext, the receiver P_{recv} can obtain a proof π that should ideally prove what message was encrypted. Formally, π is determined by the adversary (in form of a pre-stored algorithm Prove) to ensure that during the simulation, at least the shape of π matches the one of a possible real implementation. However, $\mathcal{F}_{\text{PKENO}}$ ensures that verification (via `Verify` queries) satisfies some natural and crucial requirements. Namely, an honestly (i.e., via $\mathcal{F}_{\text{PKENO}}$) generated encryption C of m cannot be proven to contain a different message $m' \neq m$. Also, honestly (i.e., via $\mathcal{F}_{\text{PKENO}}$) generated proofs are always accepted. In all cases left open by this (and in particular, if a wrong public key is used with `Verify`), the adversary is free to determine the verification outcome in order to simulate a real implementation.

Note that from the functionality's perspective, ciphertexts and proofs are merely tags and do not carry any semantics. The adversary is free to determine these tags, but the functionality takes care that decryptions and proofs are handled as ideally expected. (E.g., the ciphertext tags do not depend on the messages, or honestly generated proofs verify correctly.)

Why key management is outsourced. Also note that there are no public or secret keys in the functionality. This is unlike, e.g., in the $\mathcal{F}_{\text{PKE}}$ modelings from [6, 10, 15], which do contain a public key. This simplification is possible, since we will consider keys to be already set up, which corresponds to running a public-key encryption scheme protocol in the $\mathcal{F}_{\mathrm{PKI}}$-hybrid model (see below).

The reason *why* we opted to outsource key management into $\mathcal{F}_{\mathrm{PKI}}$ is the following: if the receiving party P_{recv} was allowed to take care of key generation on its own, then a corrupted P_{recv} could generate keys in a dishonest way. (E.g., if the public key contains a common reference string for a non-interactive zero-knowledge proof, then P_{recv} could generate this CRS along with a *trapdoor* that allows P_{recv} to cheat in the proofs. That would not have been possible with an honest generation of keys.) While our concrete scheme from Section 6 is secure even if a dishonest P_{recv} chooses its keys arbitrarily, our game-based formulation (Definition 1) guarantees nothing in that setting. Of course, an adaptation of both Definition 1 and $\mathcal{F}_{\mathrm{PKENO}}$ is possible, such that a dishonest choice of keys is reflected; we chose *not* to do so because be believe that an honest generation of keys is more realistic.

Functionality $\mathcal{F}_{\mathrm{PKENO}}$

$\mathcal{F}_{\mathrm{PKENO}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$. All session-ids *sid* used in the following are expected to be of the form $sid = (recv, sid')$, such that *sid* uniquely determines a receiving party P_{recv}.

1. Upon the first activation (no matter with which input), first:
   (a) Hand $(\texttt{KeyGen}, sid)$ to the adversary.
   (b) Receive descriptions of the plaintext domain $\mathcal{M}$, randomized algorithms $\overline{\mathsf{Encrypt}}$, $\overline{\mathsf{Prove}}$, and deterministic algorithms $\overline{\mathsf{Decrypt}}$, $\overline{\mathsf{Verify}}$ from the adversary.
   Then proceed to handle the actual query as described below.
2. Upon receiving $(\texttt{Encrypt}, sid, m)$ from some party P_j:
   (a) If $m \notin \mathcal{M}$ then output an error message to P_j.
   (b) If P_{recv} is not corrupted, set $C \leftarrow_{\mathrm{R}} \overline{\mathsf{Encrypt}}(\mathsf{length}, |m|)$. If P_{recv} is corrupted, $C \leftarrow_{\mathrm{R}} \overline{\mathsf{Encrypt}}(\mathsf{message}, m)$.
   (c) Hand C to P_j and store the tuple $(\overline{\mathsf{Encrypt}}, C, m)$. If there already is a stored tuple $(\overline{\mathsf{Encrypt}}, C, m')$ for some different message $m \neq m'$, then halt.
3. Upon receiving $(\texttt{Decrypt}, sid, C)$ from P_{recv} (and P_{recv} only):
   (a) If there is a tuple $(\overline{\mathsf{Encrypt}}, C, m')$ (for some m') stored then set $m := m'$. Otherwise, set $m \leftarrow \overline{\mathsf{Decrypt}}(C)$.
   (b) Hand m to P_{recv}.
4. Upon receiving a value $(\texttt{Prove}, sid, C)$ from P_{recv} (and P_{recv} only):
   (a) If there is a tuple $(\overline{\mathsf{Encrypt}}, C, m')$ (for some m') stored then set $m := m'$. Otherwise, set $m \leftarrow \overline{\mathsf{Decrypt}}(C)$.
   (b) Set $\pi \leftarrow_{\mathrm{R}} \overline{\mathsf{Prove}}(C, m)$ and hand π to P_{recv}. Also, store the tuple $(\overline{\mathsf{Prove}}, C, m, \pi)$; if the tag π already appears in a previously stored $\overline{\mathsf{Prove}}$ tuple then halt.
5. Upon receiving a value $(\texttt{Verify}, sid, C, m, \pi)$ from some party P_j, determine *res* as follows:
   (a) If there is a stored tuple $(\overline{\mathsf{Prove}}, C, m, \pi)$, then set $res := \texttt{accept}$.
   (b) Else, if there is a tuple $(\overline{\mathsf{Encrypt}}, C, m')$ for some $m' \neq m$, then set $res := \texttt{reject}$.
   (c) In all other cases, set $res \leftarrow \overline{\mathsf{Verify}}(C, m, \pi)$.
   Finally, hand *res* to P_j.

Fig. 1. Functionality $\mathcal{F}_{\mathrm{PKENO}}$ for public-key encryption with non-interactive openings.

Interpreting a public-key encryption scheme as a protocol. If we assume that the public/secret keys have been set up already, then, syntactically, any public-key encryption scheme $\mathsf{PKENO} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Prove}, \mathsf{Ver})$ with non-interactive opening can be interpreted as a protocol aimed at realizing $\mathcal{F}_{\mathrm{PKENO}}$. Namely, every party executes $\mathsf{Enc}_{pk}(m)$ upon $(\texttt{Encrypt}, sid, m)$ inputs, and similarly executes $\mathsf{Ver}_{pk}(C, m, \pi)$ upon $(\texttt{Verify}, sid, C, m, \pi)$ inputs. In addition, the receiving party P_{recv} (which is uniquely determined by the session id $sid = (recv, sid')$) honors $\texttt{Decrypt}$ and

$\texttt{Prove}$ inputs by using the Dec and Prove algorithms with P_{recv}'s private sk. Note that although $\mathcal{Z}$ is free to choose sid, a machine can never be invoked with two different sids (even across invocations), so there are not going to be two different secret keys that would need to be managed by one receiving party.

It remains to concretize how we imagine a trusted key setup. We do so by considering a helper functionality $\mathcal{F}_{\mathrm{PKI}}$, as depicted in Figure 2. Note that $\mathcal{F}_{\mathrm{PKE}}$ is parametrized over a key-generation algorithm Gen. That means if we consider a scheme PKENO as a protocol, we actually mean the protocol described above, run in the $\mathcal{F}_{\mathrm{PKI}}^{\mathsf{Gen}}$-hybrid model for the key-generation algorithm Gen of PKENO.

Functionality $\mathcal{F}_{\mathrm{PKI}}^{\mathsf{Gen}}$

$\mathcal{F}_{\mathrm{PKI}}^{\mathsf{Gen}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$. All session-ids sid used in the following are expected to be of the form $sid = (recv, sid')$, such that sid uniquely determines a receiving party P_{recv}. Furthermore, $\mathcal{F}_{\mathrm{PKI}}$ is parametrized over a key-generation algorithm Gen.

1. Upon the first activation (no matter with which input), first run $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{Gen}(1^k)$ to generate a public key pk along with a secret key sk.
2. Upon any input from some party P_j or the adversary, send pk to P_j. In addition, if $j = recv$, send also sk to P_j.

Fig. 2. Functionality $\mathcal{F}_{\mathrm{PKI}}$ that captures a trusted key setup.

3.2 A Game-based definition

A public-key encryption scheme with non-interactive opening is a tuple $\mathsf{PKENO} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Prove}, \mathsf{Ver})$ of algorithms such that:

- The key generation algorithm Gen takes as input a security parameter 1^k and outputs a public key pk and a secret key sk. We write $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{Gen}(1^k)$. The public key pk specifies the message space $\mathcal{M}_{pk} \leftarrow \mathsf{MSpc}(pk)$ by a mapping MSpc.
- The encryption algorithm Enc takes as input a public key pk and a message $m \in \mathcal{M}_{pk}$ and outputs a ciphertext C. We write $C \leftarrow_{\mathrm{R}} \mathsf{Enc}_{pk}(m)$.
- The deterministic decryption algorithm Dec takes as input a ciphertext C and a secret key sk. It returns a message $m \in \mathcal{M}_{pk}$ or the distinguished symbol $\perp \notin \mathcal{M}_{pk}$. We write $m \leftarrow \mathsf{Dec}_{sk}(C)$.
- The proving algorithm Prove takes as input a ciphertext C and a secret key sk. It returns a proof π. We write $\pi \leftarrow_{\mathrm{R}} \mathsf{Prove}_{sk}(C)$.
- The deterministic verification algorithm Ver takes as input a tuple (C, m, π, pk), consisting of a ciphertext C, a plaintext m, a proof π, and a public key pk. It returns a result $res \in \{\texttt{accept}, \texttt{reject}\}$. We write $res \leftarrow \mathsf{Ver}_{pk}(C, m, \pi)$.

We require that with probability overwhelming in the security parameter k, an honestly generated keypair $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{Gen}(1^k)$ satisfies the following:

- **Correctness.** For all messages $m \in \mathcal{M}_{pk}$, we have $\Pr[\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m] = 1$.
- **Completeness.** For all ciphertexts C and all possible $\pi \leftarrow \mathsf{Prove}_{sk}(C)$, we have that for $m \leftarrow \mathsf{Dec}_{sk}(C)$, algorithm $\mathsf{Ver}_{pk}(C, m, \pi)$ accepts.[4]

[4] Note that m may be $\perp$.

Definition 1 (PKENO security). *A scheme* PKENO *is* PKENO*-secure if it is* IND-CCPA *secure and satisfies computational proof soundness. We define those two below:*

IND-CCPA SECURITY. *For an adversary* A*, consider the following game:*

1. $\mathsf{Gen}(1^k)$ *outputs* (pk, sk)*. Adversary* A *is given* 1^k *and* pk*.*
2. *The adversary may make polynomially many queries to a decryption oracle* $\mathsf{Dec}_{sk}(\cdot)$ *and a proof oracle* $\mathsf{Prove}_{sk}(\cdot)$*.*
3. *At some point,* A *outputs two equal-length messages* m_0, m_1*. A bit* b *is randomly chosen and the adversary is given the challenge ciphertext* $C^* \leftarrow \mathsf{Enc}_{pk}(m_b)$*.*
4. A *may continue to query its decryption and its proof oracle, except that it may not query either with* C^**.*
5. *Finally,* A *outputs a guess* b'*.*

Denote A*'s advantage in guessing* b' *by*

$$\mathrm{Adv}^{\text{ind-ccpa}}_{\mathsf{PKENO},\mathsf{A}}(k) := \left|\Pr\left[b = b'\right] - 1/2\right| .$$

Scheme PKENO *is called* indistinguishable against chosen-ciphertext and prove attacks (IND-CCPA secure) *if for every adversary* A*,* $\mathrm{Adv}^{\text{ind-ccpa}}_{\mathsf{PKENO},\mathsf{A}}(\cdot)$ *is negligible.*

PROOF SOUNDNESS. *For an adversary* A*, consider the following game:*

1. $\mathsf{Gen}(1^k)$ *outputs* (pk, sk)*. Adversary* A *is given* 1^k *and* (pk, sk)*.*
2. *The adversary chooses a message* $m \in \{0,1\}^*$ *and gives it to an encryption oracle which returns* $C \leftarrow_{\mathrm{R}} Enc_{pk}(m)$*.*
3. *The adversary returns* (m', π')*.*

Denote A*'s probability to forge a proof by*

$$\mathrm{Adv}^{\text{snd}}_{\mathsf{PKENO},\mathsf{A}}(k) := \Pr\left[\mathsf{accept} \leftarrow \mathsf{Ver}_{pk}(C, m', \pi') \wedge m' \neq m\right] .$$

Scheme PKENO *is said to satisfy* computational proof soundness *if for every adversary* A*,* $\mathrm{Adv}^{\text{snd}}_{\mathsf{PKENO},\mathsf{A}}(\cdot)$ *is negligible.*

4 Equivalence

We will show that PKENO security is equivalent to universal composability in the sense of realizing $\mathcal{F}_{\text{PKENO}}$. The idea is simple: the guarantees that $\mathcal{F}_{\text{PKENO}}$ gives are precisely the properties that Definition 1 requires. However, there is one catch: our simulation breaks down once proofs are asked in a situation in which both sender and receiver are honest. Technically, this stems from a commitment problem the simulation runs into: if sender and receiver are honest, $\mathcal{F}_{\text{PKENO}}$ demands as secrecy guarantee that a ciphertext C in the system does not depend on the associated message m. However, if later on a *proof* is requested that C really decrypts to m, we would need to break —ironically— exactly proof soundness for a good simulation. There seems no easy way to change $\mathcal{F}_{\text{PKENO}}$ itself to prevent this: if $\mathcal{F}_{\text{PKENO}}$ behaves differently depending on whether, e.g., the receiver is corrupted or not, the sender can deduce whether the receiver is indeed corrupted or not. This however would lead to an unachievable functionality (since the receiver might be passively corrupted).

Optimistic environments. To establish equivalence of the definitions, we hence restrict to UC-environments that do not ask for proofs if both sender and receiver are uncorrupted. We call such environments *optimistic*. It is natural to assume that any larger protocol context that uses a PKENO scheme is optimistic: proofs are only requested upon conflicts, which should not happen if both parties are honest.

Theorem 1. *Say that* PKENO *is a public-key encryption scheme with non-interactive opening. Then* PKENO *is* PKENO*-secure (in the sense of Definition 1) if and only if* PKENO *(interpreted as a protocol as described in Section 3.1) securely realizes* $\mathcal{F}_{\mathrm{PKENO}}$ *in the* $\mathcal{F}_{\mathrm{PKI}}^{\mathsf{Gen}}$*-hybrid model, with respect to non-adaptive adversaries and optimistic environments.*

A formal proof will be given in Appendix A. Here, we give some intuition.

To show that universal composability implies PKENO security, attacks on PKENO's IND-CCPA and proof soundness properties must be translated into attacks on PKENO's indistinguishability from $\mathcal{F}_{\mathrm{PKENO}}$. Suppose A successfully attacks PKENO's IND-CCPA property. We build an environment $\mathcal{Z}$ that internally simulates A and the whole IND-CCPA experiment. In this, $\mathcal{Z}$ obtains decryptions and proofs via its own protocol interface (i.e., via PKENO, resp. $\mathcal{F}_{\mathrm{PKENO}}$), and the challenge message m_b is encrypted with an `Encrypt` query. In the real model, this yields a true encryption of m_b, and in the ideal model results in something independent of b by definition of $\mathcal{F}_{\mathrm{PKENO}}$. Hence the output distribution of the internally simulated A is correlated with b in the real model, and independent of b in the ideal model, which allows to distinguish. The translation of attacks on PKENO's proof soundness property works similarly.

To show that PKENO security implies universal composability, we describe a simulator $\mathcal{S}$ that, in the ideal setting with $\mathcal{F}_{\mathrm{PKENO}}$, simulates attacks performed with the dummy adversary $\tilde{\mathcal{A}}$ on PKENO. Essentially, $\mathcal{S}$ only provide algorithms for $\mathcal{F}_{\mathrm{PKENO}}$'s `Encrypt`, `Decrypt`, `Prove`, and `Verify` answers. (Of course, $\mathcal{F}_{\mathrm{PKENO}}$ enforces several rules with its answers, like proof soundness guarantees, but apart from that, $\mathcal{S}$'s algorithms determine these answers.) Algorithms for decryption, proofs, and verifications are chosen just as in the real model. (Note that $\mathcal{S}$ is free to make up a $\mathcal{F}_{\mathrm{PKI}}^{\mathsf{Gen}}$ instance on its own, so $\mathcal{S}$ knows and in fact chooses the secret keys.) The encryption algorithm for the case the sender is uncorrupted is simply yields encryptions of $1^{|m|}$ (i.e., all-one encryptions of the right length), whereas encryptions in case the sender is corrupted can be performed faithfully as in the real model (in this case, the encryption may depend on the full message, since so secrecy is guaranteed then). The proof that this simulation is sound proceeds by transforming real into ideal model, while showing that this transformation preserves $\mathcal{Z}$'s view:

1. Substituting m-encryptions with $1^{|m|}$-encryptions can be justified with PKENO's IND-CCPA property.
2. $\mathcal{F}_{\mathrm{PKENO}}$'s list-based decryption of known ciphertexts is simply an enforced correctness, which can be justified with PKENO's correctness.
3. $\mathcal{F}_{\mathrm{PKENO}}$'s verification rules can be justified with PKENO's proof soundness.

This sketches why the simulation that $\mathcal{S}$ provides is correct, and hence the theorem is proven.

Achieving full UC security. It is natural to ask whether $\mathcal{F}_{\mathrm{PKENO}}$ can be realized unconditionally, i.e., without restricting $\mathcal{Z}$. (This corresponds to composability in arbitrary protocol contexts.) As sketched above, to put up a successful simulation here, it must be possible to produce special ciphertexts (sent between an honest sender and an honest verifier) that can be opened to an *arbitrary*, a-priori unknown message. Intuitively, this seems to break proof soundness; however, this *is* possible in principle, since in the ideal model, the simulator has control over the generation of the used keypair (pk, sk). (Note that PKENO security only gives guarantees if this keypair is *honestly* generated.)

More concretely, consider the (inefficient) non-interactive zero-knowledge based scheme from the introduction. By, e.g., producing a CRS in pk with knowledge of a trapdoor, $\mathcal{S}$ is able to give fake proofs that some ciphertext really encrypts a message m. We stress that this can *not* be used to break the intuitive guarantees that $\mathcal{F}_{\mathrm{PKENO}}$ provides: $\mathcal{F}_{\mathrm{PKENO}}$ still checks that the verification of this proof succeeds only for the "right" message that is associated with a ciphertext.

5 Implementation of PKENO using IBE

5.1 Identity-based encryption

We first define syntax and required security properties of an identity-based encryption (IBE) scheme.

SYNTAX. An IBE scheme is a tuple $\mathsf{IBE} = (\mathsf{IBEgen}, \mathsf{KeyGen}, \mathsf{IBEenc}, \mathsf{IBEdec})$ of algorithms along with a family $\mathcal{M} = (\mathcal{M}_k)_k$ of message spaces such that:

- The key generation algorithm IBEgen takes as input a security parameter 1^k and outputs a public key pk and a secret key sk. We write $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{IBEgen}(1^k)$.
- The encryption algorithm IBEenc takes as input a public key pk, an identity $id \in \{0,1\}^*$ and a message $m \in \mathcal{M}_k$ and outputs a ciphertext c. We write $c \leftarrow_{\mathrm{R}} \mathsf{IBEenc}_{pk}(id, m)$.
- The deterministic decryption algorithm IBEdec takes as input a ciphertext c, an identity $id \in \{0,1\}^*$ and a user secret key $usk[id]$. It returns a message $m \in \mathcal{M}_k$ or the distinguished symbol $\perp \notin \mathcal{M}_k$. We write $m \leftarrow \mathsf{IBEdec}_{usk[id]}(c)$.
- The deterministic user secret key algorithm KeyGen takes as input an identity $id \in \{0,1\}^*$ and a secret key sk. It returns a user secret key $usk[id]$. We write $usk[id] \leftarrow \mathsf{KeyGen}_{sk}(id)$.[5]

CONSISTENCY. We require that for every honestly generated keypair $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{IBEgen}(1^k)$, for all identities $id \in \{0,1\}^*$ and messages $m \in \mathcal{M}_k$ we have $\mathsf{IBEdec}_{\mathsf{KeyGen}(sk,id)}(\mathsf{IBEenc}_{pk}(id, m)) = m$ with probability one.

Here we also require a non-standard soundness property that it is efficiently verifiable if a given user secret key $usk[id]$ was properly generated for identity id.[6] We write $\{\mathtt{accept}, \mathtt{reject}\} \leftarrow \mathsf{IBEver}_{pk}(id, usk[id])$. We require for all honestly generated keypair $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{IBEgen}(1^k)$ satisfies the following: For all identities $id \in \{0,1\}^*$ and strings $s \in \{0,1\}^*$ we have $\mathsf{IBEver}_{pk}(id, s) = \mathtt{accept}$ iff $s = usk[id]$, where $usk[id] \leftarrow \mathsf{KeyGen}_{sk}(id)$.

SECURITY. We only require a relatively weak security property, namely indistinguishability against selective-ID chosen-plaintext attacks (IND-sID-CPA) [3]. Formally, for an adversary A, consider the following game:

1. Adversary A is given 1^k and outputs a target identity id^*
2. $\mathsf{IBEgen}(1^k)$ outputs (pk, sk). Adversary A is given 1^k and pk.

[5] We can always assume the user secret key algorithm KeyGen to be deterministic. If it is not, the owner of the secret key ensures using the same randomness for each identity either by maintaining a state or by deriving the randomness using a PRF applied to the identity.

[6] It is *not* sufficient to check whether, e.g., some random encryptions decrypt correctly. A given alleged user secret key might misbehave on precisely one ciphertext.

3. The adversary may make polynomially many queries to a user secret-key oracle $\mathsf{KeyGen}_{sk}(\cdot)$, except that it may not query for id^*
4. At some point, A outputs two equal-length messages m_0, m_1. A bit b is randomly chosen and the adversary is given the challenge ciphertext $C^* \leftarrow_{\mathrm{R}} \mathsf{IBEenc}_{pk}(id^*, m_b)$.
5. A may continue to query its user secret-key oracle, except that it may not query for id^*.
6. Finally, A outputs a guess b'.

Denote A's advantage in guessing b' by

$$\mathrm{Adv}^{\text{sid-cpa}}_{\mathsf{IBE},\mathsf{A}}(k) := \left|\Pr\left[b = b'\right] - 1/2\right|.$$

Scheme IBE is called IND-sID-CPA secure if $\mathrm{Adv}^{\text{sid-cpa}}_{\mathsf{IBE},\mathsf{A}}(\cdot)$ is negligible for every PPT adversary A. We remark that there exist efficient IND-sID-CPA secure IBE schemes without random oracle [3].

5.2 From IBE to PKENO

We use an adaptation of the IBE-to-PKE transformation by Canetti, Halevi and Katz [9]. Let $\mathsf{IBE} = (\mathsf{IBEgen}, \mathsf{KeyGen}, \mathsf{IBEenc}, \mathsf{IBEdec})$ be an IBE scheme and $\mathsf{OTS} = (\mathsf{SGen}, \mathsf{SSign}, \mathsf{SVer})$ be a one-time signature scheme which we require to be strongly unforgeable against one-time attacks. (Syntax and security properties of OTS are recalled in Appendix B.1.) We construct a PKENO scheme $\mathsf{PKENO} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Prove}, \mathsf{Ver})$ as follows.

$\mathsf{Gen}(1^k)$. The key generation algorithm runs the IBE key generation algorithm $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{IBEgen}(1^k)$ and returns the key-pair (pk, sk).

$\mathsf{Enc}_{pk}(m)$. The encryption algorithm first generates a key-pair of the one-time signature scheme by running $(vk, sigk) \leftarrow_{\mathrm{R}} \mathsf{SGen}(1^k)$. Next, it IBE encrypts m with "identity" vk to obtain $c \leftarrow_{\mathrm{R}} \mathsf{IBEenc}_{pk}(vk, m)$. Finally, it signs the IBE ciphertext $\sigma \leftarrow \mathsf{SSign}_{sigk}(c)$. and returns the PKENO ciphertext $C = (vk, c, \sigma)$.

$\mathsf{Dec}_{sk}(C)$. The decryption algorithm parses C as the tuple (vk, c, σ). Next, it verifies if σ is a correct signature on c by running $\mathsf{SVer}_{vk}(c)$. If not, it returns $\perp$. Otherwise, it computes $usk[vk] \leftarrow \mathsf{KeyGen}_{sk}(vk)$ and IBE decrypts c by running $m \leftarrow \mathsf{IBEdec}_{usk[vk]}(c)$. Finally, it returns $m \in \mathcal{M}_k \cup \{\perp\}$.

$\mathsf{Prove}_{sk}(C)$. The prove algorithm parses C as the tuple (vk, c, σ). Next, it verifies if σ is a correct signature on c by running $\mathsf{SVer}_{vk}(c)$. If not, it returns $\perp$. Otherwise, it computes $usk[vk] \leftarrow \mathsf{KeyGen}_{sk}(vk)$ and returns $\pi \leftarrow usk[vk]$ as the proof.

$\mathsf{Ver}_{pk}(C, m, \pi)$. The verification algorithm parses C as the tuple (vk, c, σ). Next it verifies if σ is a correct signature on c with respect to verification key vk by running $\mathsf{SVer}_{vk}(c)$. If not, it returns `reject`. Otherwise, it checks if π is a properly generated user secret-key for "identity" vk by running $\mathsf{IBEver}_{pk}(vk, \pi)$. If not, it returns `reject`. Otherwise, it IBE decrypts c by running $\hat{m} \leftarrow \mathsf{IBEdec}_{\pi}(vk, c)$, where $\hat{m} \in \mathcal{M}_k \cup \{\perp\}$. If $\hat{m} \neq m$, it returns `reject`. Otherwise it returns `accept`.

It is easy to check that the above scheme satisfies correctness and completeness.

Theorem 2. *Assume* IBE *is* IND-sID-CPA *secure and* OTS *is* SUF-OT *secure. Then* PKENO *constructed above is* PKENO *secure.*

First note that IBE soundness directly implies perfect proof soundness of PKENO. This is since the proof algorithm makes sure that the proof $\pi = usk[vk]$ is a properly generated user secret key for the the "identity" vk from the ciphertext by running the verification algorithm. Hence by consistency of the IBE scheme the decrypted message $\hat{m}$ will always equal the real message m of the ciphertext and hence verification accepts.

Let us now give some intuition why PKENO is IND-CCPA secure. A formal proof will be given in Appendix C. Let (c^*, vk^*, σ^*) be the challenge ciphertext in the IND-CCPA security experiment. It is clear that, without any oracle queries, the value of the bit b remains hidden to the adversary. This is so because c^* is output by IBEenc which is IND-sID-CPA secure, vk^* is independent of the message, and σ^* is the result of applying the one-time signing algorithm to c^*.

We claim that decryption and proof oracle queries cannot further help the adversary in guessing the value of b. First note that a proof for some ciphertext enables the adversary to decrypt the same ciphertext without making the decryption query. It remains to consider an arbitrary proof query $(c, vk, \sigma) \neq (c^*, vk^*, \sigma^*)$ made by the adversary during the experiment. If $vk = vk^*$ then $(c, \sigma) \neq (c^*, \sigma^*)$ and the proof oracle will answer $\perp$ since the adversary is unable to forge a new valid signature σ with respect to vk^*. If $vk \neq vk^*$ then the proof query will not help the adversary since the the proof $\pi = usk[vk]$ is an IBE user secret key for the "identity" vk distinct from vk^*.

6 Direct Implementation of PKENO in Bilinear Group

6.1 Bilinear Groups and assumptions

Our schemes will be parametrized by a *pairing parameter generator*. This is an algorithm $\mathcal{G}$ that on input 1^k returns the description of an multiplicative cyclic group $\mathbb{G}$ of prime order p, where $2^k < p < 2^{k+1}$, the description of a multiplicative cyclic group $\mathbb{G}_T$ of the same order, and a non-degenerate bilinear pairing $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. We use $\mathbb{G}^*$ to denote $\mathbb{G} \setminus \{1\}$, i.e. the set of all group elements except the neutral element. The pairing has to be satisfy the following two conditions.

Non-degenerate: for all $g \in \mathbb{G}^*$, $\hat{e}(g, g) \neq 1 \in \mathbb{G}_T$.
Bilinear: for all $g \in \mathbb{G}^*$, $x, y \in \mathbb{Z}_p$, $\hat{e}(g^x, g^y) = \hat{e}(g, g)^{xy}$.

We use $\mathbb{PG} = (\mathbb{G}, \mathbb{G}_T, p, \hat{e}, g, g_T)$ as shorthand for the description of bilinear groups, where g is a generator of $\mathbb{G}$ and $g_T = \hat{e}(g, g) \in \mathbb{G}_T$. The Bilinear Decisional Diffie-Hellman (BDDH) assumption [4] states that the two distributions $(g^x, g^y, g^z, \hat{e}(g, g)^{xyz})$ and $(g^x, g^y, g^z, \hat{e}(g, g)^r)$, for $x, y, z, r \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ are indistinguishable for any adversary. More formally we define the advantage function $\mathrm{Adv}^{\mathrm{bddh}}_{\mathcal{G},\mathsf{A}}(k)$ of an adversary A as

$$|\Pr[\mathsf{A}(\mathbb{PG}, g^x, g^y, g^z, \hat{e}(g, g)^{xyz}) = 1] - \Pr[\mathsf{A}(\mathbb{PG}, g^x, g^y, g^z, \hat{e}(g, g)^r) = 1]|$$

where $\mathbb{PG} \leftarrow_{\mathrm{R}} \mathcal{G}(1^k)$ and $x, y, z, r \leftarrow_{\mathrm{R}} \mathbb{Z}_p$. We say that the Bilinear Decision Diffie-Hellman (BDDH) assumption holds relative to $\mathcal{G}$ if for every adversary A, $\mathrm{Adv}^{\mathrm{bddh}}_{\mathcal{G},\mathsf{A}}(\cdot)$ is negligible.

6.2 The PKENO scheme

Our scheme uses the "direct chosen ciphertext technique" which results in an adaptation of the IND-CCA secure PKE scheme from [5, 16]. Let $\mathsf{TCR} : \mathbb{G} \to \mathbb{Z}_p$ be a hash function that we assume to be target collision resistant [11]. Let $\mathbb{PG} \leftarrow_{\mathrm{R}} \mathcal{G}(k)$ be a pairing group that is contained in the system parameters. Let (E, D) be a symmetric encryption scheme that we assume to be chosen-ciphertext secure.[7] We assume that uses elements of the target group $\mathbb{G}_T$ as secret keys. We construct a PKENO scheme $\mathsf{PKENO} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Prove}, \mathsf{Ver})$ as follows.

$\mathsf{Gen}(1^k)$. The key generation algorithm picks random exponents $x_1, x_2, y \in Z_p$. The secret key is $sk = (x_1, x_2, y) \in \mathbb{Z}_p^3$ and the public key is $pk = (X_1, X_2, Y) \in \mathbb{G}^2 \times \mathbb{G}_T$, where

$$X_1 = g^{x_1} \in \mathbb{G}, \quad X_2 = g^{x_2} \in \mathbb{G}, \quad Y = \hat{e}(g,g)^y \in \mathbb{G}_T.$$

$\mathsf{Enc}_{pk}(m)$. The encryption algorithm first picks a random $r \in \mathbb{Z}_p$. The ciphertext is the tuple (c_1, c_2, c_3), where

$$c_1 = g^r, \quad t = \mathsf{TCR}(c_1), \quad c_2 = (X_1^t X_2)^r, \quad K \leftarrow Y^r, \quad c_3 \leftarrow \mathsf{E}_K(m)$$

$\mathsf{Dec}_{sk}(C)$. The decryption algorithm parses C as the tuple (c_1, c_2, c_3). Next, it computes $t = \mathsf{TCR}(c_1)$ and checks if $c_1^{x_1 t + x_2} \stackrel{?}{=} c_2$. If not, it returns $\perp$ meaning the ciphertext is inconsistent. Otherwise, it computes

$$K \leftarrow \hat{e}(c_1, g^y)$$

and returns $m \leftarrow \mathsf{D}_K(c_3) \in \mathcal{M} \cup \{\perp\}$.

$\mathsf{Prove}_{sk}(C)$. The prove algorithm parses C as the tuple (c_1, c_2, c_3). Next, it computes $t = \mathsf{TCR}(c_1)$ and checks if $c_1^{x_1 t + x_2} = c_2$. If not, it returns $\perp$. Otherwise, it picks $s \leftarrow_{\mathrm{R}} \mathbb{Z}_p$. The proof consists of $\pi = (d_1, d_2) \in \mathbb{G}^2$, where

$$d_1 = g^s, \quad d_2 = g^y \cdot (X_1^t X_2)^s \ . \tag{2}$$

$\mathsf{Ver}_{pk}(C, m, \pi)$. The verification algorithm parses C as the tuple (c_1, c_2, c_3) and π as the tuple (d_1, d_2). Next, it computes $t = \mathsf{TCR}(c_1)$ and checks if

$$\hat{e}(c_2, g) \stackrel{?}{=} \hat{e}(c_1, X_1^t X_2) \quad \text{and} \quad \hat{e}(g, d_2) \stackrel{?}{=} Y \cdot \hat{e}(X_1^t X_2, d_1) \ . \tag{3}$$

If one of the checks fails, it returns `reject`. Otherwise, it computes

$$\hat{K} \leftarrow \hat{e}(c_1, d_2)/\hat{e}(c_2, d_1),$$

and $\hat{m} \leftarrow \mathsf{D}_{\hat{K}}(c_3) \in \mathcal{M}_k \cup \{\perp\}$. It returns `accept` if $\hat{m} = m$ and `reject`, otherwise.

It is easy to check that the above scheme satisfies correctness and completeness.

6.3 Security

Theorem 3. *Assume the BDDH assumption holds relative to $\mathcal{G}$, TCR is a target collision-resistant hash function, and (E, D) is a chosen-ciphertext secure symmetric encryption scheme. Then PKENO constructed above is* PKENO *secure.*

[7] A symmetric encryption scheme is chosen-ciphertext secure if the encryptions of two adversarially-chosen messages under a random hidden key K remain indistinguishable even relative to a decryption oracle. We refer to [11] for a formal definition.

The proof of IND-CCPA security is similar to the one from [5, 16] and omitted here.

We verify proof soundness. Fix a key-pair and let $C = (c_1 = g^r, c_2 = (X_1^t X_2)^r, c_3 = \mathsf{E}_K(m))$ be a proper encryption of a message m, where $K = Y^r$ is the symmetric key used for encrypting m. Now consider the verification algorithm run with C, a message $m' \neq m$ and an arbitrary proof $\pi' = (d'_1, d'_2)$. The right check of (3) implies that $\pi' = (d'_1, d'_2)$ is a properly generated proof of the form (2), for some $s \in \mathbb{Z}_p$ and for $t = \mathsf{TCR}(c_1)$. Hence, for the symmetric key $\hat{K}$ we have

$$\hat{K} = \hat{e}(c_1, d'_2)/\hat{e}(c_2, d'_1) = \hat{e}(g^r, g^y \cdot (X_1^t X_2)^s)/\hat{e}((X_1^t X_2)^r), g^s) = Y^r = K$$

By consistency of the symmetric scheme the recovered message $\hat{m} = \mathsf{D}_K(c_3)$ equals $m \neq m'$, hence verification always outputs `reject`.

References

[1] Michael Backes, Markus Dürmuth, Dennis Hofheinz, and Ralf Küsters. Conditional reactive simulatability. In Eugene Asarin, Dieter Gollmann, Jan Meier, and Andrei Sabelfeld, editors, *Computer Security, Proceedings of ESORICS 2006*, Lecture Notes in Computer Science, pages 424–443. Springer-Verlag, 2006. Extended version online available at `http://eprint.iacr.org/2006/132.ps`.

[2] Michael Backes and Birgit Pfitzmann. Limits of the cryptographic realization of Dolev-Yao-style XOR. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *Computer Security, Proceedings of ESORICS 2005*, number 3679 in Lecture Notes in Computer Science, pages 178–196. Springer-Verlag, 2005. Online available at `http://eprint.iacr.org/2005/220.ps`.

[3] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 223–238. Springer-Verlag, Berlin, Germany, May 2004.

[4] Dan Boneh and Matthew K. Franklin. Identity based encryption from the Weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.

[5] Xavier Boyen, Qixiang Mei, and Brent Waters. Direct chosen ciphertext security from identity-based techniques. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 05*, pages 320–329. ACM Press, November 2005.

[6] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001. Full version online available at `http://www.eccc.uni-trier.de/eccc-reports/2001/TR01-016/revisn01.ps`.

[7] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. IACR ePrint Archive, January 2005. Online available at `http://eprint.iacr.org/2000/067.ps`.

[8] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology, Proceedings of CRYPTO 2001*, number 2139 in Lecture Notes in Computer Science, pages 19–40. Springer-Verlag, 2001. Full version online available at `http://eprint.iacr.org/2001/055.ps`.

[9] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 207–222. Springer-Verlag, Berlin, Germany, May 2004.

[10] Ran Canetti, Hugo Krawczyk, and Jesper B. Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, *Advances in Cryptology, Proceedings of CRYPTO 2003*, number 2729 in Lecture Notes in Computer Science, pages 565–582. Springer-Verlag, 2003. Full version online available at `http://eprint.iacr.org/2003/174.ps`.

[11] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[12] Ivan Damgård and Rune Thorbek. Non-interactive proofs for integer multiplication. In Moni Naor, editor, *Advances in Cryptology, Proceedings of EUROCRYPT 2007*, number 4515 in Lecture Notes in Computer Science, pages 412–429. Springer-Verlag, 2007. Full version online available at `http://eprint.iacr.org/2007/086`.

[13] Anupam Datta, Ante Derek, John C. Mitchell, Ajith Ramanathan, and Andre Scredrov. Games and the impossibility of realizable ideal functionality. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Proceedings of TCC 2006*, number 3876 in Lecture Notes in Computer Science, pages 360–379. Springer-Verlag, 2006. Online available at `http://eprint.iacr.org/2005/211.pdf`.

[14] Craig Gentry. Practical identity-based encryption without random oracles. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 445–464. Springer-Verlag, Berlin, Germany, May / June 2006.
[15] Dennis Hofheinz, Jörn Müller-Quade, and Rainer Steinwandt. On modeling IND-CCA security in cryptographic protocols. *Tatra Mountains Mathematical Publications*, 2005. 14 pages, to be published.
[16] Eike Kiltz. Chosen-ciphertext security from tag-based encryption. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 581–600. Springer-Verlag, Berlin, Germany, March 2006.
[17] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *44th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2003*, pages 394–403. IEEE Computer Society, 2003. Full version online available at `http://eprint.iacr.org/2003/141.ps`.
[18] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*. ACM Press, May 1990.
[19] Jesper B. Nielsen. *On Protocol Security in the Cryptographic Model.* PhD thesis, University of Aarhus, 2003. Online available at `http://www.brics.dk/~buus/jbnthesis.ps.gz`.
[20] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. Cryptology ePrint Archive, Report 2007/279, 2007. `http://eprint.iacr.org/`.
[21] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127. Springer-Verlag, Berlin, Germany, May 2005.

A Proof of Theorem 1

"$\Leftarrow$" For the easier "if" direction, say that PKENO is not PKENO-secure. That means that there is either a successful IND-CCPA adversary attacking PKENO, or an adversary that successfully attacks PKENO's proof soundness (or both).

Case 1: If PKENO is not IND-CCPA. Suppose $\mathrm{Adv}^{\text{ind-ccpa}}_{\mathsf{PKENO},\mathsf{A}}(k)$ is non-negligible for some adversary A. Then the following environment $\mathcal{Z}$ successfully distinguishes PKENO from $\mathcal{F}_{\mathrm{PKENO}}$. In a nutshell, $\mathcal{Z}$, running in the real model with the dummy adversary $\tilde{\mathcal{A}}$, runs the IND-CCPA experiment with an internal simulation of A. More concretely, $\mathcal{Z}$ proceeds as follows:

1. Ask[8] the UC adversary $\mathcal{A}$ for P_{recv}'s public key pk. In the real model, $\mathcal{A}$ can obtain pk through $\mathcal{F}_{\mathrm{PKI}}$, and in the ideal model, the simulator can freely make up some pk.
2. Run A with input pk, relaying Dec and Prove oracle queries to P_{recv} in form of `Decrypt` and `Prove` queries.
3. When A outputs two messages m_0, m_1, choose $b \in \{0,1\}$ uniformly and ask P_{recv} for an encryption C^* of m_b under pk. Hand C^* as challenge ciphertext to A.
4. Continue to relay A's Dec and Prove oracle queries to P_{recv}, except for queries $C = C^*$.
5. When A outputs a guess b' for b, halt with output $b \oplus b'$.

By construction, $\mathcal{Z}$ faithfully simulates the IND-CCPA experiment for A in the real model, so

$$\left|\Pr\left[\mathrm{REAL}_{\Pi,\tilde{\mathcal{A}},\mathcal{Z}}(k,z)=1\right]-1/2\right| = \mathrm{Adv}^{\text{ind-ccpa}}_{\mathsf{PKENO},\mathsf{A}}(k).$$

However, in the ideal model, regardless of the simulator $\mathcal{S}$, the encryption C^* is by definition of $\mathcal{F}_{\mathrm{PKENO}}$ independent of b: The challenge ciphertext C^* is generated by $\mathcal{F}_{\mathrm{PKENO}}$ as $C^* = \mathsf{Encrypt}(\ell)$ for $\ell = |m_0| = |m_1|$, and no decryptions of C^* are allowed after C^* is set up. Hence $\Pr\left[\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{PKENO}},\mathcal{S},\mathcal{Z}}(k,z)=1\right] = 1/2$, and so

$$\left|\Pr\left[\mathrm{REAL}_{\Pi,\tilde{\mathcal{A}},\mathcal{Z}}(k,z)=1\right]-\Pr\left[\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{PKENO}},\mathcal{S},\mathcal{Z}}(k,z)=1\right]\right| = \mathrm{Adv}^{\text{ind-ccpa}}_{\mathsf{PKENO},\mathsf{A}}(k)$$

is non-negligible by assumption on A.

[8] The session id *sid* can be chosen arbitrarily, e.g., such that $P_{recv} = P_1$.

CASE 2: IF PKENO DOES NOT HAVE SOUND PROOFS. Suppose $\mathrm{Adv}^{\mathrm{snd}}_{\mathsf{PKENO},\mathsf{A}}(k)$ is non-negligible for some adversary A. Again, we construct an environment $\mathcal{Z}$ that successfully distinguishes PKENO from $\mathcal{F}_{\mathrm{PKENO}}$. Similarly to the IND-CCPA case, $\mathcal{Z}$ simulates the proof soundness game for A. Concretely, $\mathcal{Z}$ proceeds as follows:

1. Ask the (dummy) adversary to corrupt P_{recv} and look up P_{recv}'s keypair (pk, sk).
2. Internally run A with input (pk, sk).
3. When A wants to encrypt a message m, ask P_j (for some $j \neq recv$ to encrypt this message under pk. Hand the obtained ciphertext C back to A.
4. When A finally outputs a message $m' \neq m$ along with a proof π', ask P_j to verify this proof with a $(\mathtt{Verify}, sid, pk, C, m', \pi')$ query. Output 1 iff P_j accepts the proof.

In the real model with dummy adversary $\tilde{\mathcal{A}}$, this perfectly mimics the proof soundness experiment of Definition 1. Hence $\Pr\left[\mathrm{REAL}_{\Pi,\tilde{\mathcal{A}},\mathcal{Z}}(k,z) = 1\right]$ is non-negligible by assumption on A. In the ideal model, however, $\Pr\left[\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z) = 1\right] = 0$ since $\mathcal{F}_{\mathrm{PKENO}}$ ensures that C cannot be proven as an encryption of m', when C was previously generated as an encryption of $m \neq m'$.

"$\Rightarrow$" For the "only if" direction, assume that PKENO is PKENO-secure in the sense of Definition 1. To show that PKENO securely realizes $\mathcal{F}_{\mathrm{PKENO}}$, it suffices to give a simulator $\mathcal{S}$ such that for the dummy adversary $\tilde{\mathcal{A}}$ and every environment $\mathcal{Z}$,

$$\Pr\left[\mathrm{REAL}_{\Pi,\tilde{\mathcal{A}},\mathcal{Z}}(k,z) = 1\right] \approx \Pr\left[\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z) = 1\right]. \tag{4}$$

So consider the simulator $\mathcal{S}$ that acts as follows. Upon a $\mathtt{KeyGen}$ request from $\mathcal{F}_{\mathrm{PKENO}}$, it generates a keypair via $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{Gen}(1^k)$, specifies the plaintext space $\mathcal{M}_{pk} \leftarrow \mathsf{MSpc}(pk)$, and sends $\mathcal{M}_{pk}$ along with algorithms $\overline{\mathsf{Encrypt}}$, $\overline{\mathsf{Decrypt}}$, $\overline{\mathsf{Prove}}$, $\overline{\mathsf{Verify}}$ defined as follows back to $\mathcal{F}_{\mathrm{PKENO}}$.

– $\overline{\mathsf{Encrypt}}(\mathsf{length}, \ell)$ outputs $\mathsf{Enc}_{pk}(1^{\ell})$.
– $\overline{\mathsf{Encrypt}}(\mathsf{message}, m)$ outputs $\mathsf{Enc}_{pk}(m)$.
– $\overline{\mathsf{Decrypt}}(C)$ outputs $\mathsf{Dec}_{sk}(C)$.
– $\overline{\mathsf{Prove}}(C)$ outputs $\mathsf{Prove}_{sk}(C)$.
– $\overline{\mathsf{Verify}}(C, m, \pi)$ outputs $\mathsf{Ver}_{pk}(C, m, \pi)$.

Requests by $\mathcal{Z}$ to corrupt a party and deliver the party's internal state must be answered only initially, since we consider non-adaptive adversaries only. Hence, $\mathcal{S}$ can answer with a fresh party state. Also, since PKENO, interpreted as a protocol, sends no messages, requests by $\mathcal{Z}$ to report messages can be answered trivially as well. Requests by $\mathcal{Z}$ to let a corrupted party communicate with $\mathcal{F}_{\mathrm{PKI}}$ are answered by an internal simulation of $\mathcal{F}_{\mathrm{PKI}}$ (which uses $\mathcal{S}$'s own (pk, sk) keypair). That means that the only chance for $\mathcal{Z}$ to observe a difference between the ideal setting with $\mathcal{F}_{\mathrm{PKENO}}$ and $\mathcal{S}$, and the real setting with PKENO and $\tilde{\mathcal{A}}$ lies in the answers to input queries (i.e., answers to $\mathtt{Encrypt}$, $\mathtt{Decrypt}$, $\mathtt{Prove}$, and $\mathtt{Verify}$ queries).

Fix an arbitrary environment $\mathcal{Z}$. Without loss of generality, we may assume that $\mathcal{Z}$ asks $\tilde{\mathcal{A}}$ to initially corrupt P_{recv} either always or never.

IF P_{recv} IS CORRUPTED. We go through the input queries that $\mathcal{Z}$ can make one by one, to see when a different behavior in real and ideal model can happen:

1. $\mathtt{Encrypt}$ queries. Since P_{recv} is corrupted, $\mathtt{Encrypt}$ queries are answered by $\mathcal{F}_{\mathrm{PKENO}}$ using $\overline{\mathsf{Encrypt}}(\mathsf{message}, m)$ invocations, which produce authentic real encryptions of m. Hence real and ideal model $\mathtt{Encrypt}$ queries are answered identically *unless* $\overline{\mathsf{Encrypt}}$ produces a ciphertext that

has been generated before for *another* message (in which case $\mathcal{F}_{\mathrm{PKENO}}$ halts by construction to avoid ambiguities). Fortunately, since we required perfect correctness from $\mathcal{F}_{\mathrm{PKENO}}$, this cannot happen.

2. `Decrypt` queries. Note that $\mathcal{F}_{\mathrm{PKENO}}$ uses $\overline{\mathsf{Decrypt}}$ to decrypt ciphertexts (which gives the same behavior as in the real model), except if the ciphertext to be decrypted has been generated before by $\mathcal{F}_{\mathrm{PKENO}}$. In that latter case, perfect correctness of $\mathcal{F}_{\mathrm{PKENO}}$ guarantees that real and ideal model do not differ here.
3. `Prove` queries. By construction, `Prove` queries are answered by $\mathcal{F}_{\mathrm{PKENO}}$ and $\mathcal{S}$ exactly as in the real model.
4. `Verify` queries. Here real and ideal answers do not differ, unless one of the following happens:
   α For a proof π generated by $\mathcal{F}_{\mathrm{PKENO}}$ and the corresponding message m, we have $\mathtt{reject} \leftarrow \mathsf{Ver}_{pk}(C, m, \pi)$ (which differs from $\mathcal{F}_{\mathrm{PKENO}}$'s output `accept`).
   β For a ciphertext C generated by $\mathcal{F}_{\mathrm{PKENO}}$ and the corresponding message m, we have $\mathtt{accept} \leftarrow \mathsf{Ver}_{pk}(C, m, \pi)$ (which differs from $\mathcal{F}_{\mathrm{PKENO}}$'s output `reject`).
   Now α can never happen by $\mathcal{F}_{\mathrm{PKENO}}$'s completeness, and the probability for β to happen is negligible, which can shown by a straightforward reduction to $\mathcal{F}_{\mathrm{PKENO}}$'s proof soundness property.

To summarize, $\mathcal{Z}$'s view in real and ideal model differs only negligibly if P_{recv} is corrupted. We will show that $\mathcal{Z}$'s view in real and ideal model differs only negligibly, which shows the overall claim.

If P_{recv} is not corrupted. We will successively change the real model with PKENO and $\tilde{\mathcal{A}}$ into the ideal model with $\mathcal{F}_{\mathrm{PKENO}}$ and $\mathcal{S}$, making in each step changes that only cause a negligible difference in $\mathcal{Z}$'s view. We start with the real model.

Substituting encryptions. Our first change affects the answers to `Encrypt` queries, which we change from encryptions of m to encryptions of $1^{|m|}$ (just like $\mathcal{F}_{\mathrm{PKENO}}$, resp. $\overline{\mathsf{Encrypt}}$ would produce them). Decryptions of ciphertexts that are generated from `Encrypt` queries are hardwired to yield the original message m (instead of $1^{|m|}$). The claim that this changes $\mathcal{Z}$'s view only negligibly can be reduced to PKENO's IND-CCPA property with a standard hybrid argument.[9]

Handling encryption collisions. The second change concerns again `Encrypt`. We let all parties halt when an `Encrypt` query for a message m results in a ciphertext C, where C already has been returned as an `Encrypt` encryption for a message $m' \neq m$. Note that this refers to an event in which $\mathcal{F}_{\mathrm{PKENO}}$ would have halted. Unfortunately, we cannot rely on PKENO's correctness here to show that this results in a negligible change: unlike in the case P_{recv} is corrupted, we are dealing with collisions of ciphertexts that *really* may be Enc encryptions of the same message 1^ℓ (if $\ell = |m| = |m'|$). However, we *can* show that two ciphertexts C, C' of the same message 1^ℓ are equal only with negligible probability with a reduction to PKENO's IND-CCPA property. (The idea is as follows: if $\mathcal{Z}$ manages to provoke a collision of ciphertexts as above, then this collision can be used to distinguish encryptions of 1^ℓ from encryptions of, say, 0^ℓ.)

The remaining queries. Note that these changes modify not only `Encrypt`, but also `Decrypt` queries to be answered exactly like ideal $\mathcal{F}_{\mathrm{PKENO}}$ queries. By construction, `Prove` queries are already answered just as with $\mathcal{F}_{\mathrm{PKENO}}$ (and $\mathcal{S}$). It remains to take care of `Verify` queries. With completely analogous reasoning as above (where P_{recv} is assumed corrupted), we can change the treatment of `Verify` queries to the ideal $\mathcal{F}_{\mathrm{PKENO}}$ way without causing a non-negligible change in $\mathcal{Z}$'s view.

[9] We stress that it is crucial here that we required $\mathcal{Z}$ to never ask for proofs if both sender and receiver are honest. Hence the reduced IND-CCPA adversary never asks for a proof for a challenge message.

Summarizing, we have changed the real setting into the ideal setting without changing $\mathcal{Z}$'s view non-negligibly.

B Standard definitions

B.1 One-Time Signatures

A signature scheme is a tuple $\mathsf{OTS} = (\mathsf{SGen}, \mathsf{SSign}, \mathsf{SVer})$ of PPT algorithms such that:

- The randomized key generation algorithm SGen takes as input a security parameter 1^k and outputs a verification key vk and a signing key $sigk$. We write $(sigk, vk) \leftarrow_{\mathrm{R}} \mathsf{SGen}(1^k)$.
- The randomized signing algorithm SSign takes as input a signing key $sigk$ and a message $m \in \{0,1\}^*$ and outputs a signature σ. We write $\sigma \leftarrow_{\mathrm{R}} \mathsf{SSign}_{sigk}(m)$.
- The deterministic verification algorithm SVer takes as input a verification key vk and a signature σ. It returns `accept` or `reject`. We write $\{\texttt{accept}, \texttt{reject}\} \leftarrow \mathsf{SVer}_{vk}(\sigma)$.

For an adversary F, consider the following game:

1. $\mathsf{SGen}(1^k)$ outputs $(sigk, vk)$. Adversary F is given 1^k and vk.
2. The adversary may make one query m to a signing oracle $\mathsf{SSign}_{sigk}(\cdot)$ which results in the signature σ.
3. Finally, F outputs a message m^* and a signature σ^*.

Denote F's advantage by

$$\mathrm{Adv}^{\text{suf-ot}}_{\mathsf{OTS},\mathsf{F}}(k) := \Pr[\mathsf{SVer}_{vk}(m^*, \sigma^*) = \texttt{accept} \wedge (m, \sigma) \neq (m^*, \sigma^*)].$$

OTS is called strongly unforgeable against one-time attacks (SUF-OT secure) if for all adversaries F $\mathrm{Adv}^{\text{suf-ot}}_{\mathsf{OTS},\mathsf{F}}(\cdot)$ is negligible.

C Proof of Theorem 2

Let A be an IND-CCPA adversary against PKENO. We show how to construct an IND-sID-CPA adversary B against IBE and an SUF-OT adversary F against OTS (both with about the same time complexity as A) such that

$$\mathrm{Adv}^{\text{ind-ccpa}}_{\mathsf{PKENO},\mathsf{A}}(k) \leq \mathrm{Adv}^{\text{sid-cpa}}_{\mathsf{IBE},\mathsf{B}}(k) + \mathrm{Adv}^{\text{suf-ot}}_{\mathsf{OTS},\mathsf{F}}(k).$$

We start by describing B that interacts with the IND-sID-CPA security game against IBE as follows.

1. On input 1^k, B picks $(sigk^*, vk^*) \leftarrow_{\mathrm{R}} \mathsf{SGen}(1^k)$ and returns vk^* as the challenge identity.
2. Adversary B inputs pk and runs A on pk.
3. As already pointed out it is sufficient for B to answer A's proof queries. Let $C = (vk, C, \sigma)$ be such a proof query made by A. First, B checks if the signature σ is correct and rejects if not. Next, if $vk = vk^*$ then B terminates and reports failure. Next, B queries its $\mathsf{KeyGen}_{sk}(\cdot)$ oracle for the user secret key $usk[vk]$ of "identity vk" and returns the proof $\pi = usk[vk]$ to A.
4. B inputs the two messages (m_0, m_1) from A and submits them to its own experiment obtaining a challenge ciphertext c^* for identity vk^*. It returns $C = (vk^*, c^*, \sigma^*)$ to A, where $\sigma^* \leftarrow \mathsf{SSign}_{sigk}(c^*)$.

5. B continues answering A's proof queries, as above.
6. Finally A returns a bit b' which B outputs to its own IBE experiment.

We denote Fail the event that B reports failure in the above simulation. Let WinA and WinB be the events that A and B win their security experiment in the above simulation. Note that B perfectly simulates A's view in the IND-CCPA experiment unless Fail happens and we have $\Pr[\mathsf{WinB} \wedge \neg\mathsf{Fail}] = \Pr[\mathsf{WinA} \wedge \neg\mathsf{Fail}]$. Hence

$$|\mathsf{Pr}\left[\mathsf{WinA}\right] - \mathsf{Pr}\left[\mathsf{WinB}\right]| \leq \mathsf{Pr}\left[\mathsf{Fail}\right],$$

and we get

$$\begin{aligned}\mathrm{Adv}^{\text{ind-ccpa}}_{\mathsf{PKENO},\mathsf{A}} &= |\Pr[\mathsf{WinA}] - 1/2| \\ &\leq |\Pr[\mathsf{WinB}] - 1/2| + \mathsf{Pr}\left[\mathsf{Fail}\right] \\ &= \mathrm{Adv}^{\text{sid-cpa}}_{\mathsf{IBE},\mathsf{B}} + \mathsf{Pr}\left[\mathsf{Fail}\right].\end{aligned}$$

We now give an adversary F such that

$$\mathrm{Adv}^{\text{suf-ot}}_{\mathsf{OTS},\mathsf{F}} \geq \Pr[\mathsf{Fail}]. \tag{5}$$

Adversary F carries out the above simulation with the following two changes. First, it generates $(pk, sk) \leftarrow_{\mathrm{R}} \mathsf{IBEgen}(1^k)$ itself hence being able to simulate the surrounding IBE IND-sID-CPA security experiment. Second, instead of generating $(vk^*, sigk^*)$ itself, it only inputs vk^* from the SUF-OT experiment against OTS. Adversary F only has to make one call to the signing oracle to obtain the signature σ^* on c^* with respect to vk^*. Consider the case that Fail happens and let $(vk = vk^*, C, \sigma)$ be the corresponding proof query. Since $(C, \sigma) \neq (C^*, \sigma^*)$, σ is a valid strong forgery on C with respect to vk^*. This implies (5) and completes the proof.