Public-Key Encryption with Non-interactive Opening and its Applications

Contributors Inc.

ECRYPT MAYA WG1

Abstract. We reconsider proof soundness for public-key encryption with non-interactive opening (PKENO) proposing new definitions that are stronger than those considered so far. We give a taxonomy of all definitions and demonstrate them to be satisfiable. Moreover, we show that PKENO is equivalent to another primitive: robust non-interactive threshold public-key encryption (TPKE). Using this result, we construct an efficient PKENO scheme that does not make use of pairings.

Keywords: public-key encryption, non-interactive proofs, random oracle model, standard model.

1 Introduction

Public-key encryption allows a receiver Bob to generate a pair of private and a public keys (sk_B, pk_B). Using pk_B, anyone can encrypt messages that can only be decrypted by Bob thanks to the knowledge of sk_B. The primitive *public-key encryption with non-interactive opening* (PKENO) allows Bob to prove to a verifier Alice what the result of decrypting a given ciphertext C produced under pk_B is. By using PKENO, Bob can do so convincingly and without interaction. More precisely, Bob runs a proving algorithm Prove on inputs its secret key sk_B and the intended ciphertext C, thereby generating a proof π. On the other hand, Alice runs a verification algorithm Ver on inputs Bob's public key pk_B, ciphertext C, a plaintext m, and an opening proof π. The verification algorithm outputs 1 if C was indeed an encryption of m, and 0 otherwise. An interesting feature of PKENO is that Bob can also convince Alice of the fact that a given ciphertext C is *invalid*, i.e. it is rejected by the decryption algorithm.

PKENO turns out to be a useful primitive for protocol design, as we illustrate with the following examples.

Secure Message Transmission with PKENO. One of the classical ways to realize secure message transmission with non-repudiation in a public-key setting is to let the sender encrypt the message and then sign the ciphertext, i.e. the so-called *encrypt-then-sign* paradigm [**?**] in which the resulting message is of the form $\mathtt{Sign}(sk_s, \mathsf{Enc}(pk_r, pk_s||m))$ with sk_s being the signing key of the sender and pk_r the encryption key of the receiver. If the sender uses some IND-CCA2 secure PKE scheme the receiver will only be able to provide a non-repudiable proof for the origin of the received ciphertext, i.e. the sender's signature. Replacing PKE with PKENO would lead to the additional ability of the receiver to prove the origin for the decrypted message. Interestingly, this is equivalent to using the *sign-then-encrypt* paradigm [**?**] where the message is first signed and then encrypted together with the signature using some IND-CCA2 secure PKE scheme. Moreover, using PKENO in the encrypt-then-sign construction would additionally provide non-repudiable proof of the destination through the corresponding soundness proof of the PKENO scheme.

PKENO in Distributed Key Generation. The idea behind distributed key generation (DKG) in [?], which improves upon [?], is to let users agree on a public key pk such that the corresponding secret key sk is distributed amongst the users $U_1, \ldots, U_n$ in form of secret shares $s_1, \ldots, s_n$. In these protocols sk has usually the form $f(sk_1, \ldots, sk_n)$ for some function f and individual secrets sk_i chosen by U_i. During the protocol execution each U_i must securely transmit a share ss_i of sk_i accompanied with the corresponding signature σ_i to all other users. The verifiability of the process allows each receiver U_i to check whether some received share ss_j from U_j is valid or not. If not this user can complain against the sender U_j by publicizing (ss_j, σ_j). In this context PKENO offers an alternative realization, i.e. instead of signing the share ss_i and securely transmitting it together with the signature σ_i to the users it would suffice for each sender to transmit authenticated PKENO-based encryption of ss_i. In this way any receiver U_i can complain about U_j by presenting the decrypted share ss_j together with the PKENO proof for the valid decryption.

PKENO and Commitment Schemes. PKENO implies a computationally secure commitment scheme in which a (possibly malicious) sender generates (sk, pk). In order to commit the sender simply computes the PKENO ciphertext $C = \mathsf{Enc}(pk, m)$. The decommitment is given by (m, π) where $\pi = \mathsf{Prove}(sk, C)$. The computational hiding property follows from the semantic security whereas the computational binding property relies on the computational proof soundness of the secure PKENO scheme.

Group Signatures. The most common way to achieve anonymity in group signatures is the following: a member first produces a (non-anonymous) signature which he then encrypts under the opener's public key while adding a proof of well-formedness. If the encrypted message contains information to identify the signer then a group signature is easily *opened* by decrypting the ciphertext it contains.

In the security model for dynamic group signatures by Bellare et al. [?], the opening authority is required to give a *proof* that it traced the correct user. Using PKENO rather than plain encryption enables the opener to do so. In the game defining anonymity of group signatures, an adversary is provided an opening oracle which opens a signature of the adversary's choice and outputs a proof of correctness of opening. IND-CCPA (Definition 1) of the employed PKENO scheme (together with simulation-sound zero knowledge of the proof of well-formedness) guarantees that an adversary cannot distinguish signatures of two different users.

Non-frameability, another security notion in [?], states that even a collusion comprising the issuer (who enrolls members with the group) and the opener cannot produce a *proof* that a signature opens to a user who did not produce it. Note that non-frameability does not rely on the security of the PKENO scheme but rather on that of the underlying signatures: it is required that the message decrypted by the opener can only have been generated by the accused user.

1.1 Our Contributions

Difficulty of Building PKENO. [?] showed that a PKENO can be built out of any Identity-Based Encryption (IBE) scheme. However IBE is a very restricted cryptographic functionality, so it is interesting to see whether PKENO can be realized without resorting to all the functionalities provided by IBE (such as key extraction). In [?] the authors mentioned that PKENO can be also based upon a seemingly weaker primitive, called *public-key encryption with witness-recovering decryption* (PKEWR) [?]. In a PKEWR scheme, the receiver – Bob is

able to recover the random coins r used to encrypt a ciphertext C. Damgård *et al.* proposed to use the r as the proof, and verification proceeds by re-encrypting $C' = \mathsf{Enc}(pk_B, m; r)$ and checking whether $C = C'$, accepting/rejecting the proof accordingly. However, this approach can only be guaranteed to be sound for *valid* ciphertexts, i.e. ciphertexts that have been output by the encryption algorithm. But, in particular invalid ciphertexts do not fall into this set. As a consequence, as it was also stated elsewhere in [**?**], for invalid ciphertexts the concept "the coins used to construct C" might be a not well defined concept. Indeed, we show in Section ?? how the straightforward construction of PKENO out of PKEWE fails to provide security. This then motivates the search for new generic/concrete constructions for PKENO.

Non-Interactive Threshold Encryption Implies PKENO. Somewhat surprisingly, we show that *robust non-interactive threshold public-key encryption* (TPKE) implies PKENO. TPKE schemes distribute the ability to decrypt among several parties. The private decryption key is shared among n servers such that at least t servers are needed for decryption. If the *combiner* wishes to decrypt some ciphertext C, it sends C to the decryption servers. After receiving at least t partial decryption shares from the servers, the combiner is able to reconstruct the plaintext from these shares. A *robust* TPKE [**?**,**?**] provides the additional property that whenever the decryption of valid ciphertexts fails, the combiner can sieve out bad decryption shares and reveal the identity of the server having sent an invalid partial decryption.

Stronger Soundness Definitions. The main motivation for the introduction of PKENO was protocol design: some player sends a message to Bob securely by encrypting it under his public key. If Bob finds out (possibly at some later time) that the message is somehow "invalid", he can convince other participants of this fact without getting back to the (eventually) dishonest sender. *Proof soundness* ensures that Bob can do so convincingly; in particular, it states that if a ciphertext C encrypts a message m, then Bob cannot make a proof for C being an encryption of a different message m'. In the game that formally defines this security notion, the challenger produces a pair public and secret key, hands it to the adversary, who outputs a message of which he receives an encryption C. The adversary wins if he outputs a different message and makes a valid proof that this was the opening.

In real-world applications, the keys are usually chosen by the users themselves; which in the case of proof soundness would be the adversary. It seems thus natural to let *the adversary* choose the keys in the security experiment to reflect this fact. We define two stronger flavors of proof soundness, where the first one is analogous to the original definition given by [**?**], but letting the adversary choose his keys; the second one states that no adversary can find a public key, a ciphertext and two messages and a proof of opening for each of them.

Note that the strengthening of proof soundness also makes sense for the other applications given above; in particular for group signatures, the opener could choose his opening key and add corresponding information to the public parameters. Strong proof soundness then guarantees non-frameability even in this setting.

2 Preliminaries

In this section we review the definitions and tools required to present our results. We start by fixing some notation.

2.1 Notation

If x is a string then $|x|$ denotes its length, while if S is a set then $|S|$ denotes its size. If k is a natural number, then 1^k denotes the string of k ones. If S is a set then $s_1, \ldots, s_n \xleftarrow{\$} S$ denotes the operation of picking n elements s_i of S independently and uniformly at random. We write $\mathcal{A}(x, y, \ldots)$ to indicate that $\mathcal{A}$ is an algorithm with inputs $x, y, \ldots$ and by $z \leftarrow \mathcal{A}(x, y, \ldots)$ we denote the operation of running $\mathcal{A}$ with inputs $(x, y, \ldots)$ and letting z be the output. We use the abbreviation PPT to refer to a probabilistic polynomial-time algorithm [**?**].

2.2 Public Key Encryption Scheme with Non-interactive Opening

A PKENO scheme $\mathsf{PKENO} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Prove}, \mathsf{Ver})$ is a tuple of five PPT algorithms:

- Gen is a probabilistic algorithm taking as input a security parameter 1^k. It returns a public key pk and a secret key sk. The public key includes the description of the set of plaintexts $\mathcal{M}_{pk}$.
- Enc is a probabilistic algorithm taking as inputs a public key pk and a message $m \in \mathcal{M}_{pk}$. It returns a ciphertext C.
- Dec is a deterministic algorithm that takes as inputs a ciphertext C and a secret key sk. It returns a message $m \in \mathcal{M}_{pk}$ or the special symbol $\perp$ meaning that C is invalid.
- Prove is a probabilistic algorithm taking as inputs a ciphertext C and a secret key sk. It returns a proof π.
- Ver is a deterministic algorithm taking as inputs a public key pk, a ciphertext C, a plaintext m and a proof π. It returns a result $res \in \{0, 1\}$ meaning accepted and rejected proof respectively. In particular, $1 \leftarrow \mathsf{Ver}(pk, C, \perp, \pi)$ must be interpreted as the verifier being convinced that C is an invalid ciphertext.

Correctness requires that for an honestly generated key pair $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$ the following holds:

- For all messages $m \in \mathcal{M}_{pk}$ we have $\Pr\left[\mathsf{Dec}\big(\,\mathsf{Enc}(pk, m)\,\big) = m\right] = 1$.
- For all ciphertexts C we have

$$\Pr\left[\, 1 \leftarrow \mathsf{Ver}\big(\, pk, C, \mathsf{Dec}(sk, C), \mathsf{Prove}(sk, C)\,\big)\,\right] = 1\ .$$

Security of PKENO is defined by indistinguishability under chosen-ciphertext and prove attacks (IND-CCPA) and proof soundness [**?**,**?**]. We formally define both notions and propose strengthened definitions for proof soundness.

Definition 1 (IND-CCPA security). *Let us a consider the following game between a challenger and an adversary $\mathcal{A}$:*

Setup *The challenger runs $\mathsf{Gen}(1^k)$ and gives pk to $\mathcal{A}$.*
Phase 1 *The adversary issues queries of the form:*
a) decryption query to an oracle $\mathsf{Dec}(sk, \cdot)$;
b) proof query to an oracle $\mathsf{Prove}(sk, \cdot)$.
These queries may be asked adaptively, that is, they may depend on the answers to previous queries.
Challenge *At some point, $\mathcal{A}$ outputs two equal-length messages $m_0, m_1 \in \mathcal{M}_{pk}$. The challenger chooses a random bit β and returns $C^\star \leftarrow \mathsf{Enc}(pk, m_\beta)$.*

Phase 2 *As Phase 1, except that neither decryption nor proof queries on $C^\star$ are allowed.*
Guess *The adversary $\mathcal{A}$ outputs a guess $\beta' \in \{0,1\}$. The adversary wins the game if $\beta = \beta'$.*

Define $\mathcal{A}$'s advantage as $\mathsf{Adv}^{\text{ind-ccpa}}_{\mathsf{PKENO},\mathcal{A}}(1^k) = \left|\Pr[\beta' = \beta] - 1/2\right|$. *A scheme* PKENO *is called* indistinguishable against chosen-ciphertext and prove attacks (IND-CCPA secure) *if for every PPT adversary $\mathcal{A}$,* $\mathsf{Adv}^{\text{ind-ccpa}}_{\mathsf{PKENO},\mathcal{A}}(\cdot)$ *is negligible.*

Definition 2 (Proof Soundness). *Consider the following game between a challenger and an adversary $\mathcal{A}$:*

Stage 0 $\mathsf{Gen}(1^k)$ *outputs a pair of keys (pk, sk). Adversary $\mathcal{A}$ is given (pk, sk).*
Stage 1 *The adversary chooses a message $m \in \mathcal{M}_{pk}$.*
Stage 2 *The challenger computes $C \leftarrow \mathsf{Enc}(pk, m)$ and gives it to $\mathcal{A}$ which returns (m', π').*

$\mathcal{A}$'s advantage is defined as the probability

$$\mathbf{Adv}^{\text{proof-snd}}_{\mathsf{PKENO},\mathcal{A}}(1^k) := \Pr\left[1 \leftarrow \mathsf{Ver}(pk, C, m', \pi') \wedge m' \neq m\right] .$$

A scheme PKENO *is* proof sound *if for every PPT adversary $\mathcal{A}$ its advantage is negligible.*

Below, we strengthen the previous definition of proof soundness [**?**,**?**] by giving the adversary full control over the choice of the public key.

Definition 3 (Strong Proof Soundness). *Consider the following game between a challenger and an adversary $\mathcal{A}$:*

Stage 1 $\mathcal{A}(1^k)$ *outputs a public key pk and a message $m \in \mathcal{M}_{pk}$*
Stage 2 *The challenger computes $C \leftarrow \mathsf{Enc}(pk, m)$ and gives it to $\mathcal{A}$, which returns (m', π').*

$\mathcal{A}$'s advantage is defined as the probability

$$\mathbf{Adv}^{\text{s-proof-snd}}_{\mathsf{PKENO},\mathcal{A}}(1^k) := \Pr\left[1 \leftarrow \mathsf{Ver}(pk, C, m', \pi') \wedge m' \neq m\right] .$$

A scheme PKENO *is* strongly proof sound *if for every PPT adversary $\mathcal{A}$ its advantage is negligible.*

An alternative strong notion of proof soundness (with adversarially chosen keys) follows the idea that for any ciphertext one can only find one valid message-proof pair. We call this the committing property:

Definition 4 (Committing Property). *A scheme* PKENO *is* strongly committing *if for every adversary $\mathcal{A}$, that on input 1^k outputs $(pk, C, m, \pi, m', \pi')$, the following probability is negligible:*

$$\mathbf{Adv}^{\text{s-com}}_{\mathsf{PKENO},\mathcal{A}}(1^k) := \Pr\left[1 \leftarrow \mathsf{Ver}(pk, C, m, \pi) \wedge 1 \leftarrow \mathsf{Ver}(pk, C, m', \pi') \wedge m \neq m'\right] .$$

We show in Appendix A that strong proof soundness implies proof soundness and that the committing property and strong soundness are somewhat incomparable. The following shows that Definitions 3 and 4 are actually achievable—by a practical scheme.

Theorem 1. *The "repaired scheme" in* [**?**] *is strongly proof-sound and strongly committing.*

The proof is deferred to Appendix B where the mentioned scheme can also be found.

2.3 Robust Non-Interactive Threshold Public-Key Encryption

Non-interactive threshold public-key encryption schemes (TPKE), as formalized in [**?**], distributes the ability to decrypt among several parties. The private decryption key is shared among n servers such that at least t servers are needed for decryption. If the *combiner* wishes to decrypt some ciphertext C, it sends C to the decryption servers. After receiving at least t partial decryption shares from the servers, the combiner is able to reconstruct the plaintext from these shares. A *robust* TPKE [**?**,**?**] provides the additional property that whenever the decryption of valid ciphertexts fails, the combiner can sieve out bad decryption shares and reveal the identity of the server having sent an invalid partial decryption.

Syntax. We use the same syntax as Boneh-Boyen-Halevi [**?**] and Shoup-Gennaro [**?**] for (robust) non-interactive threshold public key encryption (TPKE). Formally, such a robust TPKE scheme $\mathsf{TPKE} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{ShareDecrypt}, \mathsf{ShareVerify}, \mathsf{Combine})$ consists of the following algorithms:

- $\mathsf{Setup}(n, t, 1^\lambda)$ takes as input a security parameter 1^λ and integers $t, n \in \mathbb{N}$ (with $1 \leq t \leq n$) denoting the number of decryption servers n and the decryption threshold t. It outputs a triple $(\mathsf{PK}, \mathbf{VK}, \mathbf{SK})$, where PK is the public key, $\mathbf{SK} = (\mathsf{SK}_1, \ldots, \mathsf{SK}_n)$ is a vector of n private key shares and $\mathbf{VK} = (\mathsf{VK}_1, \ldots, \mathsf{VK}_n)$ is the corresponding vector of verification keys. Decryption server i is given the share (i, SK_i) that allows to derive decryption shares for any ciphertext. For each $i \in \{1, \ldots, n\}$, the verification key VK_i is used to check the validity of decryption shares generated using SK_i.
- $\mathsf{Encrypt}(\mathsf{PK}, M)$ given a public key PK and a plaintext, this randomized algorithm outputs a ciphertext C.
- $\mathsf{ShareDecrypt}(\mathsf{PK}, i, \mathsf{SK}_i, C)$ on input of a public key PK, a ciphertext C and a private key share (i, SK_i), this (possibly randomized) algorithm outputs either a decryption share $\mu_i = (i, \hat{\mu}_i)$, or a special symbol $(i, \perp)$.
- $\mathsf{ShareVerify}(\mathsf{PK}, \mathsf{VK}_i, C, \mu_i)$ takes as input PK, the verification key VK_i, a ciphertext C and a purported decryption share $\mu_i = (i, \hat{\mu}_i)$. It outputs either `valid` or `invalid`. In the former case, μ_i is said to be a valid decryption share.
- $\mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, \{\mu_1, \ldots, \mu_t\})$ given PK, $\mathbf{VK}$, C and a set of t valid decryption shares $\{\mu_1, \ldots, \mu_t\}$, this algorithm outputs a plaintext M or $\perp$.

It is required that the consistency of PK with $\mathbf{VK}$ be publicly checkable. Namely, for any t-subset V of $\mathbf{VK}$, there must be an efficient algorithm allowing to make sure that V is a valid set of verification keys w.r.t. PK.

Correctness. For any $(\mathsf{PK}, \mathbf{VK}, \mathbf{SK})$ generated by $\mathsf{Setup}(n, t, 1^\lambda)$, it is required that

1. for any ciphertext C, if $\mu_i = \mathsf{ShareDecrypt}(\mathsf{PK}, i, \mathsf{SK}_i, C)$, where SK_i is the i^{th} private key share in $\mathbf{SK}$, then $\mathsf{ShareVerify}(\mathsf{PK}, \mathsf{VK}_i, C, \mu_i) = \mathtt{valid}$;
2. if C is the output of $\mathsf{Encrypt}(\mathsf{PK}, M)$ and $S = \{\mu_1, \ldots, \mu_t\}$ is a set of decryption shares such that $\mu_i = \mathsf{ShareDecrypt}(\mathsf{PK}, i, \mathsf{SK}_i, C)$ for t distinct private key shares in $\mathbf{SK}$, then $\mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, S) = M$.

The security of TPKE schemes is defined via two properties. The first one is the usual notion of chosen-ciphertext security for public key encryption while the other one is termed

consistency of decryptions.

CHOSEN-CIPHERTEXT SECURITY. Security against chosen-ciphertext attacks is defined by the following game where the challenger is faced with a static adversary that decides which servers it wants to corrupt upfront. Both parties take integers n, t and the security parameter λ as input.

Definition 5 (IND-TCCA security). *Let us consider the following game between a challenger and an adversary $\mathcal{A}$:*

Initialization *The adversary outputs a set $S \subset \{1, \ldots, n\}$ of $t-1$ decryption servers that it wishes to corrupt.*

Setup *The challenger runs $\mathsf{Setup}(n, t, 1^\lambda)$ to obtain a triple $(\mathsf{PK}, \mathbf{VK}, \mathbf{SK} = (\mathsf{SK}_1, \ldots, \mathsf{SK}_n))$. The adversary is given $\mathsf{PK}, \mathbf{VK}$ and decryption shares (j, SK_j) for indices $j \in S$.*

Query stage 1 *The adversary $\mathcal{A}$ makes decryption queries (C, i), where $i \in \{1, \ldots, n\}$. The challenger replies with $\mu_i = \mathsf{ShareDecrypt}(\mathsf{PK}, i, \mathsf{SK}_i, C)$.*

Challenge *The adversary outputs plaintexts M_0, M_1 and obtains $C^\star = \mathsf{Encrypt}(\mathsf{PK}, M_b)$ for a random bit $b \xleftarrow{\$} \{0, 1\}$ chosen by the challenger.*

Query stage 2 *The adversary $\mathcal{A}$ makes new decryption queries (C, i) under the natural restriction that $C \neq C^\star$. The challenger responds as in stage 1.*

Guess *The adversary outputs $b' \in \{0, 1\}$ and wins if $b' = b$.*

Define $\mathcal{A}$'s advantage as $\mathsf{Adv}^{\text{ind-tcca}}_{\mathsf{TPKE},\mathcal{A}}(1^k) := |\Pr[b' = b] - 1/2|$. A scheme TPKE is called indistinguishable against threshold-chosen-ciphertext *attacks (IND-TCCA) if for every PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\text{ind-tcca}}_{\mathsf{TPKE},\mathcal{A}}(1^k)$ is negligible.*

DECRYPTION CONSISTENCY. Shoup and Gennaro define decryption consistency using a similar game to the one above.

Definition 6 (Decryption Consistency). *Let us consider the following game between a challenger and an adversary $\mathcal{A}$:*

Initialization, Setup, and Query stage 1 *Defined as in Definition 5.*

Guess *The adversary outputs a valid-looking ciphertext C and two sets of decryption shares $S = \{\mu_1, \ldots, \mu_t\}$, $S' = \{\mu'_1, \ldots, \mu'_t\}$ of size t.*

Define $\mathcal{A}$'s advantage $\mathsf{Adv}^{\text{dec-con}}_{\mathsf{TPKE},\mathcal{A}}(1^k)$ as the probability that the following conditions hold:

1. *All decryption shares in S and S' are valid decryption shares w.r.t. the verification key $\mathbf{VK}$ and the ciphertext C.*
2. *S and S' each contain decryption shares from t distinct servers.*
3. $\mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, S) \neq \mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, S')$.

A TPKE scheme is decryption consistent *if for every PPT adversary $\mathcal{A}$ its advantage is negligible.*

In upcoming sections, we will consider somewhat stronger flavors of decryption consistency. In the first one, we require the adversary's advantage to remain negligible in a modified game where the challenger reveals PK and *all* decryption shares $\mathsf{SK}_1, \ldots, \mathsf{SK}_n$ in the setup phase.

Definition 7 (Decryption Consistency with Known Secret Keys). *Let us consider the following game between a challenger and an adversary $\mathcal{A}$:*

Setup *The challenger runs* $\mathsf{Setup}(n, t, 1^\lambda)$ *to obtain a triple* $(\mathsf{PK}, \mathbf{VK}, \mathbf{SK} = (\mathsf{SK}_1, \ldots, \mathsf{SK}_n))$ *and sends* $(\mathsf{PK}, \mathbf{VK}, \mathbf{SK})$ *to the adversary* $\mathcal{A}$.

Output $\mathcal{A}$ *generates a valid-looking ciphertext* C *and two distinct* t*-element sets* $S = \{\mu_1, \ldots, \mu_t\}$ *and* $S' = \{\mu'_1, \ldots, \mu'_t\}$ *of decryption shares.*

Define $\mathcal{A}$*'s advantage* $\mathsf{Adv}^{\text{s-dec-con}}_{\mathsf{TPKE},\mathcal{A}}(1^k)$ *as the probability that the following conditions hold:*

1. *All decryption shares in* S *and* S' *are valid decryption shares w.r.t. the verification key* $\mathbf{VK}$ *and the ciphertext* C.
2. S *and* S' *each contain decryption shares from* t *distinct servers.*
3. $\mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, S) \neq \mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, S')$.

A TPKE scheme is decryption consistent with known secret keys *if for every PPT adversary* $\mathcal{A}$ *the advantage* $\mathsf{Adv}^{\text{s-dec-con}}_{\mathsf{TPKE},\mathcal{A}}(1^k)$ *is negligible.*

We strengthen the definition in that we let the adversary choose the keys on its own.

Definition 8 (Strong Decryption Consistency). *A TPKE scheme is* strong decryption consistent *if for every PPT adversary* $\mathcal{A}$ *the advantage in a game that is similar to the above one is negligible, except that* $\mathcal{A}$ *is allowed to generate consistent encryption/verification keys* $(\mathsf{PK}, \mathbf{VK})$ *on her own* without *having to publish the vector of decryption shares* $\mathbf{SK}$.

3 Robust TPKE and PKENO Are Equivalent

3.1 Robust TPKE Implies PKENO

Let $\mathsf{TPKE} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{ShareDecrypt}, \mathsf{ShareVerify}, \mathsf{Combine})$ be a robust TPKE scheme providing chosen-ciphertext security and strong decryption consistency. We can turn it into a secure PKENO scheme $\mathsf{PKENO} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Prove}, \mathsf{Ver})$ as follows:

- $\mathsf{Gen}(1^\lambda)$ Choose arbitrary integers $t, n \in \mathbb{N}$ such that $1 \leq t \leq n$ and run $\mathsf{Setup}(n, t, 1^\lambda)$ to obtain $(\mathsf{PK}, \mathbf{VK} = (\mathsf{VK}_1, \ldots, \mathsf{VK}_n), \mathbf{SK} = (\mathsf{SK}_1, \ldots, \mathsf{SK}_n))$. The key pair (sk, pk) for PKENO is defined as $pk = (\mathsf{PK}, \mathbf{VK}, n, t)$, $sk = \mathbf{SK} = (\mathsf{SK}_1, \ldots, \mathsf{SK}_n)$. The plaintext (resp. ciphertext) space of PKENO is the plaintext (resp. ciphertext) space of TPKE.
- $\mathsf{Enc}(pk, M)$ To encrypt M under $pk = (\mathsf{PK}, \mathbf{SK}, n, t)$, compute $C = \mathsf{Encrypt}(\mathsf{PK}, M)$.
- $\mathsf{Dec}(sk, C)$ To decrypt C, conduct the following steps:
  1. For $i = 1, \ldots, t$, compute $\mu_i = \mathsf{ShareDecrypt}(\mathsf{PK}, i, \mathsf{SK}_i, C)$.
  2. If there exists $j \in \{1, \ldots, t\}$ such that $\mu_j = (j, \perp)$ return $\perp$.
  3. Otherwise, given valid shares $S = \{\mu_1, \ldots, \mu_t\}$, return $M = \mathsf{Combine}(\mathsf{PK}, \mathsf{VK}, C, S)$.
- $\mathsf{Prove}(sk, C)$ To generate a proof for the ciphertext C, parse sk as $(\mathsf{SK}_1, \ldots, \mathsf{SK}_n)$ and do the following.
  1. For $i = 1, \ldots, t$, compute $\mu_i = \mathsf{ShareDecrypt}(\mathsf{PK}, i, \mathsf{SK}_i, C)$.
  2. Return the set of decryption shares $\pi = \{\mu_1, \ldots, \mu_t\}$.
- $\mathsf{Ver}(pk, C, M, \pi)$ parse pk as $(\mathsf{PK}, \mathbf{VK}, n, t)$ and π as a set of shares $\{\mu_1, \ldots, \mu_t\}$.
  1. Return 0 if π contains less than t shares or if $(\mathsf{VK}_1, \ldots, \mathsf{VK}_t)$ is inconsistent with PK.
  2. Return 0 if there exists $j \in \{1, \ldots, t\}$ such that $\mathsf{ShareVerify}(\mathsf{PK}, \mathsf{VK}_j, C, \mu_j) = \texttt{invalid}$. Otherwise (*i.e.*, if all shares are valid), return 1 if $M = \mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, \{\mu_1, \ldots, \mu_t\})$ and 0 otherwise.

Remark 1. In the concrete constructions shown in Section 4.1, we set $n = t = 1$ for efficiency reasons (so that the consistency check between PK and $(\mathsf{VK}_1, \ldots, \mathsf{VK}_t)$ becomes trivial at step 1 of the verification algorithm). We do so with the Shoup-Gennaro [**?**] and the Boneh-Boyen-Halevi [**?**] TPKE schemes. See Section 4.

Theorem 2. *Robust non-interactive TPKE implies PKENO.*

The statement of the above theorem is implied by Lemmas 1 and 2.

Lemma 1. *The above* PKENO *system provides IND-CCPA security if the underlying* TPKE *scheme is IND-TCCA secure*

Proof. Let $\mathcal{A}$ be an IND-CCPA adversary against PKENO. We show how it simply implies a chosen-ciphertext adversary $\mathcal{B}$ against the underlying TPKE.

$\mathcal{B}$ starts by choosing $S = \{1, \ldots, t-1\}$ as the set of decryption servers to corrupt and obtains $(\mathsf{PK}, \mathbf{VK})$ as well as $((1, \mathsf{SK}_1), \ldots, (t-1, \mathsf{SK}_{t-1}))$ from her own challenger. The PKENO adversary $\mathcal{A}$ is supplied with a public key $pk = (\mathsf{PK}, \mathbf{VK}, n, t)$ and starts making decryption and proving queries. Whenever $\mathcal{A}$ queries a proof for some ciphertext C, $\mathcal{B}$ is able to compute $\mu_i = \mathsf{ShareDecrypt}(\mathsf{PK}, i, \mathsf{SK}_i, C)$ for $i = 1, \ldots, t-1$ since she knows $\mathsf{SK}_1, \ldots, \mathsf{SK}_{t-1}$. To obtain the missing decryption share, $\mathcal{B}$ queries her challenger to reveal $\mu_t = \mathsf{ShareDecrypt}(\mathsf{PK}, t, \mathsf{SK}_t, C)$, which allows constructing a valid proof $\pi = \{\mu_1, \ldots, \mu_t\}$ as long as TPKE provides correctness. It is not hard to see that $\mathcal{A}$'s decryption queries can be dealt with exactly in the same way: instead of revealing the set $\{\mu_1, \ldots, \mu_t\}$, $\mathcal{B}$ returns the output of $\mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, \{\mu_1, \ldots, \mu_t\})$.

At the challenge step, $\mathcal{A}$ outputs equal-length messages M_0, M_1 that are transmitted to $\mathcal{B}$'s challenger. The latter replies with a challenge TPKE ciphertext $C^\star$, which $\mathcal{B}$ relays to $\mathcal{A}$. In the second stage, $\mathcal{A}$ is allowed to make further decryption/proof queries. Since these never involve the challenge ciphertext $C^\star$, $\mathcal{B}$ is always able to answer them by invoking her own challenger as in the first phase. The game ends with $\mathcal{A}$ outputting a bit $b \in \{0, 1\}$, which is also $\mathcal{B}$'s result. It is straightforward to observe that, if $\mathcal{A}$ is successful, so is $\mathcal{B}$. □

Lemma 2. *The* PKENO *scheme is sound (resp. strongly committing) if* TPKE *satisfies decryption consistency with known secret keys (resp. strong decryption consistency).*

Proof. We first show that, if an adversary $\mathcal{A}$ defeats the soundness of PKENO in the sense of definition 2, there exists an adversary $\mathcal{B}$ breaking the decryption consistency with known secret keys in TPKE with the same advantage.

Namely, our adversary $\mathcal{B}$ obtains PK, $\mathbf{VK}$ and $\mathbf{SK} = (\mathsf{SK}_1, \ldots, \mathsf{SK}_n)$ from her challenger. The weak soundness adversary $\mathcal{A}$ then receives $pk = (\mathsf{PK}, \mathbf{VK}, n, t)$, $sk = \mathbf{SK}$. In stage 1 of the game, $\mathcal{A}$ chooses a plaintext m that $\mathcal{B}$ encrypts using the public key PK of TPKE. Upon receiving the resulting ciphertext $C = \mathsf{Encrypt}(\mathsf{PK}, m)$, $\mathcal{A}$ attempts to produce a pair (m', π') such that $\mathsf{Ver}(pk, C, m', \pi') = 1$ and $m' \neq m$. Since π' is a valid proof, it can necessarily be parsed as a set $\{\mu'_1, \ldots, \mu'_t\}$ of valid decryption shares. The correctness property of TPKE implies that, since $\mathcal{B}$ knows $\mathbf{SK} = (\mathsf{SK}_1, \ldots, \mathsf{SK}_n)$, she must be able to generate another set $\pi = \{\mu_1, \ldots, \mu_t\}$ of decryption shares such that $m = \mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, \{\mu_1, \ldots, \mu_t\})$. It comes that the sets π and π' are valid t-sets of decryption shares that break the decryption consistency with known secret keys of TPKE.

Proving that the strong decryption consistency of TPKE implies the strong committing property of PKENO is fairly straightforward: from a strong committingness adversary $\mathcal{A}$, we

immediately obtain a strong decryption consistency adversary $\mathcal{B}$ that outputs whatever $\mathcal{A}$ comes up with. □

3.2 PKENO Implies TPKE

Theorem 3 (Informal). *PKENO implies robust non-interactive TPKE (by extending the non-robust non-interactive TPKE construction of Dodis and Katz [**?**]).*

4 PKENO Concrete Constructions

In this section we describe some concrete schemes obtained from the transformation in Section 3.1.

4.1 PKENO without Pairings in the Random Oracle Model

This section shows that, in the random oracle model, the TPKE scheme of Shoup and Gennaro (more precisely, the cryptosystem dubbed TDH2 in [**?**]) satisfies decryption consistency in the statistical sense and thus gives rise to a strongly sound PKENO system. The scheme makes use of a cyclic group $\mathbb{G} = \langle g \rangle$ of prime order p and hash functions $H_0 : \mathbb{G} \to \{0,1\}^l$; $H_1, H_2 : \{0,1\}^l \times \mathbb{G}^4 \to \mathbb{Z}_p$ that are modeled as random oracles.

- $\mathsf{Setup}(n, t, 1^\lambda)$ chooses $x \xleftarrow{\$} \mathbb{Z}_p$, $\bar{g} \xleftarrow{\$} \mathbb{G}$ and sets $h = g^x$. The public key PK includes $g, h, \bar{g}$, the hash functions H_0, H_1, H_2 and the description of the plaintext space $\mathcal{M}_{pk} = \{0,1\}^l$. The vector of secret key shares is obtained as $\mathbf{SK} = (x_1, \ldots, x_n)$, where $x_i = F(i)$ for $i = 1, \ldots, n$ and $F[X] \in \mathbb{Z}_p[X]$ is a random $(t-1)$-degree polynomial such that $F(0) = x$. The verification key is defined to be $\mathbf{VK} = (h_1, \ldots, h_n) = (g^{x_1}, \ldots, g^{x_n})$. The consistency of any t-subset $V \subset \mathbf{VK}$ with PK can be verified by checking that $h = \prod_{j \in V} h_i^{\Lambda_{i,V}}$, using Lagrange coefficients $\Lambda_{i,V}$.
- $\mathsf{Encrypt}(\mathsf{PK}, m)$ to encrypt a message $m \in \{0,1\}^l$, the algorithm chooses $r, s \xleftarrow{\$} \mathbb{Z}_p$, it sets $K = h^r$ and computes

  $$c = H_0(h^r) \oplus m,\ u = g^r,\ w_1 = g^s,\ \bar{u} = \bar{g}^r,\ \overline{w}_1 = \bar{g}^s,\ e_1 = H_1(c, u, w_1, \bar{u}, \overline{w}_1),\ f_1 = s + r e_1.$$

  Let us notice that $(w_1, \overline{w}_1, f_1)$ constitutes a non-interactive zero-knowledge proof of equality of discrete logarithms $\log_g u = \log_{\bar{g}} \bar{u}$ [**?**]. The ciphertext is $C = (c, u, \bar{u}, e_1, f_1)$.
- $\mathsf{ShareDecrypt}(\mathsf{PK}, i, \mathsf{SK}_i, C)$ given C and the private key share $\mathsf{SK}_i = x_i$, the algorithm first checks whether $e_1 = H_1(c, u, w_1, \bar{u}, \overline{w}_1)$, where $w_1 = g^{f_1}/u^{e_1}$, $\overline{w}_1 = \bar{g}^{f_1}/\bar{u}^{e_1}$. If this test is not satisfied, it returns $(i, \perp)$. Otherwise, it computes $K_i = u^{x_i}$, $w_{i,2} = g^{s_i}$, $\overline{w}_{i,2} = u^{s_i}$, with $s_i \xleftarrow{\$} \mathbb{Z}_p^*$, and returns $\mu_i = (i, (K_i, e_{i,2}, f_{i,2}))$, where

  $$e_{i,2} = H_2(K_i, w_{i,2}, \overline{w}_{i,2}),\ f_{i,2} = s_i + x_i e_{i,2}.$$

  Recall that $(w_{i,2}, \overline{w}_{i,2}, e_{i,2}, f_{i,2})$ constitutes a non-interactive zero-knowledge proof of equality of discrete logarithms $\log_g h = \log_u K_i$.
- $\mathsf{ShareVerify}(\mathsf{PK}, \mathsf{VK}_i, C, \mu_i)$ on input of PK, $\mathsf{VK}_i = h_i = g^{x_i}$, $C = (c, u, \bar{u}, e_1, f_1)$ and $\mu_i = (i, \hat{\mu}_i)$, where $\hat{\mu}_i = (K_i, e_{i,2}, f_{i,2})$, this algorithm performs the following tests:

Test 1: if $e_1 = H_1(c, u, w_1, \bar{u}, \overline{w}_1)$, where $w_1 = g^{f_1}/u^{e_1}$, $\overline{w}_1 = \bar{g}^{f_1}/\bar{u}^{e_1}$
Test 2: if $e_{i,2} = H_2(K_i, w_{i,2}, \overline{w}_{i,2})$, where $w_{i,2} = g^{f_{i,2}}/h_i^{e_{i,2}}$, $\overline{w}_{i,2} = u^{f_{i,2}}/K_i^{e_{i,2}}$.

If these tests are both correct, it returns 1. If Test 1 fails, it outputs 1 iff $\hat{\mu}_i = \perp$. In any other case, it outputs 0.

– $\mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, S)$ parses S as $\{\mu_1, \ldots, \mu_t\}$ and μ_i as $(i, \hat{\mu}_i)$ for each i. If $\hat{\mu}_i = \perp$ for $i = 1, \ldots, t$, it returns $\perp$ (which indicates that C is invalid). Otherwise, it parses $\hat{\mu}_i$ as $(K_i, e_{i,2}, f_{i,2})$ for each i and computes $K = \prod_{j \in S} K_i^{\Lambda_{i,S}}$ using Lagrange coefficients $\Lambda_{i,S}$. It finally returns the plaintext $m = c \oplus H_0(K)$.

The IND-TCCA security of the scheme was proved in [**?**] under the Decision Diffie-Hellman assumption. It is not hard to see that it also satisfies decryption consistency in the statistical sense (although a weaker flavor of consistency was originally considered in [**?**]).

Theorem 4. *The Shoup-Gennaro TPKE system satisfies strong decryption consistency in the random oracle model.*

Proof. Since the consistency of PK with any t-subset of $\mathbf{VK} = (g^{x_1}, \ldots, g^{x_n})$ is publicly verifiable, we assume that $\mathcal{A}$ does not come up with an inconsistent pair $(\mathsf{PK}, \mathbf{VK})$. Let us consider the two sets $S = \{\mu_1, \ldots, \mu_t\}$ and $S' = \{\mu'_1, \ldots, \mu'_t\}$ produced by $\mathcal{A}$. We first observe that we cannot have $\mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, S) = \perp$ and $\mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, S') \neq \perp$ or vice versa. Indeed, since S and S' both contain valid shares, it holds that either the ciphertext $C = (c, u, \bar{u}, e_1, f_1)$ comprises an invalid proof (e_1, f_1) (in which case verification test 1 fails for all shares and $\mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, S) = \mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, S') = \perp$) or all shares $\mu_i = (i, \hat{\mu}_i) \in S$, $\mu'_j = (j, \hat{\mu}_j) \in S'$ are such that $\hat{\mu}_i \neq \perp$, $\hat{\mu}'_j \neq \perp$. We are thus left with the latter case $\perp \neq \mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, S) \neq \mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, S') \neq \perp$.

Then, it is easy to see that $\mathcal{A}$ can only break the statistical decryption consistency in the event that one of the two t-sets $S = \{\mu_1, \ldots, \mu_t\}$ and $S' = \{\mu'_1, \ldots, \mu'_t\}$ contains an ill-formed decryption share $\mu_i = (i, (K_i, e_{i,2}, f_{i,2}))$ (such that $\log_u(K_i) \neq \log_g(h_i)$) that nevertheless happens to satisfy the verification test $e_{i,2} = H_2(K_i, g^{f_{i,2}}/h_i^{e_{i,2}}, u^{f_{i,2}}/K_i^{e_{i,2}})$.

For any share $(i, (K_i, e_{i,2}, f_{i,2}))$ such that $\log_u(K_i) \neq \log_g(h_i)$, let us consider random oracle queries of the form $H_2(K_i, w_{i,2}, \overline{w}_{i,2})$, for some $(w_{i,2}, \overline{w}_{i,2}) \in \mathbb{G}^2$. For each such random oracle query, there exists exactly one pair $(e_{i,2}, f_{i,2}) \in (\mathbb{Z}_p)^2$ such that $w_{i,2} = g^{f_{i,2}}/h_i^{e_{i,2}}$ and $\overline{w}_{i,2} = u^{f_{i,2}}/K_i^{e_{i,2}}$ (since $\log_u(K_i) \neq \log_g(h_i)$). Since H_2 is a random oracle, its output accidentally hits e_i with probability $1/p$. If q_{H_2} denotes the number of H_2-queries, the overall probability that an invalid share passes the verification test is at most tq_{H_2}/p. □

The result of section 3.1 immediately implies that the Shoup-Gennaro TPKE scheme can be turned into a strongly committing PKENO in the random oracle model.

```
Benoit: There is no need to explicitly include the long proof from
scratch and I commented it out in the .tex file. Likewise, is it OK
with you to replace the following theorem (that relates to PKENO)
with the above one?

David : Yes ! :-)
```

Theorem 5. *The scheme above is* strongly committing *in the Random Oracle Model.*

Proof. Assume towards contradiction that an PPT adversary outputs with non-negligible probability a tuple $(pk, C, m, \pi, m', \pi')$ such that $\mathsf{Ver}(pk, C, m, \pi) = \mathsf{Ver}(pk, C, m', \pi') = 1$ and $m \neq m'$ holds. Note that both verifications rely on the same $pk = (g, h, \bar{g})$ and $C = (c, u, \bar{u}, e_1, f_1)$.

We can now distinguish between two cases: either $m = \perp$ and $m' \in \{0,1\}^l$ or $m, m' \in \{0,1\}^l$.

- Case 1: If one of the messages is $m = \perp$, claiming that the ciphertext C is invalid, the first verification test $e_1 \stackrel{?}{=} H_1(c, u, w_1, \bar{u}, \overline{w}_1)$ must validate as false. However, for the second message $m' \in \{0,1\}^l$ the same test must validate as true in order to pass the entire verification. Since the check $e_1 \stackrel{?}{=} H_1(c, u, w_1, \bar{u}, \overline{w}_1)$ solely depends on pk and C and not on the message, only one of both verifications can return '1'.
- Case 2: In the second case we have $m, m' \in \{0,1\}^l$, i.e., for both messages and their proofs π, π' all tests in the verification algorithm must be passed. Assume that C is a proper encryption of m, and $\pi = (K, e_2, f_2)$ is a corresponding valid proof. Thus, π includes a valid NIZK that $\log_g h = \log_u K$. Now consider the verification for the same C, pk but for a distinct message $m' \neq m$ and an arbitrary proof $\pi' = (K', e_2', f_2')$. The final test $m' = c \oplus H_0(K')$ implies that $K' \neq K$. But then, π' must contain a NIZK that $\log_g h = \log_u K'$ and thus that $\log_u K' = \log_u K$ which contradicts the soundness of the equality of discrete logarithm protocol [**?**].

□

4.2 PKENO Based on the Decision Linear Assumption

Recently, Arita and Tsurudome [**?**] described an efficient way to thresholdize the decryption algorithm of Kiltz's tag-based encryption scheme [**?**] using bilinear maps to achieve robustness. We point out that their scheme readily yields another PKENO with strong soundness. The security proof of the resulting scheme is in the standard model. One of its advantages is that it can be used in CCA2-anonymous group signatures that rely on *linear encryption* [**?**]. For instance, it can be used to obtain simpler and more efficient proofs of correct opening in Groth's fully secure group signatures [**?**]: such a proof only consists of two group elements and its verification only entails two pairing evaluations, which is significantly cheaper than checking a pairing-based non-interactive witness indistinguishable proof as in [**?**].

- $\mathsf{Setup}(n, t, 1^\lambda)$ Choose $U, V \stackrel{\$}{\leftarrow} \mathbb{G}$, $x, y \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ and set $X = g^x$, $Y = g^y$. The public key PK consists of g, h, U, V, X, Y, the description of a strongly secure one-time signature $\Sigma = (\mathcal{G}, \mathcal{S}, \mathcal{V})$ and the description of the plaintext space $\mathcal{M}_{pk} = \mathbb{G}$. The vector $\mathbf{SK}$ of private key shares is obtained as $\mathbf{SK} = ((x_1, y_1), \ldots, (x_n, y_n))$, with $x_i = F_1(i)$, $y_i = F_2(i)$ for all $i \in \{1, \ldots, n\}$, using random $(t-1)$-degree polynomials $F_1[X], F_2[X] \in \mathbb{Z}_p[X]$ subject to $F_1(0) = x^{-1}$ and $F_2(0) = y^{-1}$. The vector of verification key then consists of $\mathbf{VK} = ((v_1, w_1), \ldots, (v_n, w_n)) = ((g^{x_1}, g^{y_1}), \ldots, (g^{x_n}, g^{y_n}))$. Again, the consistency of PK with any t-subset $V \subset \mathbf{VK}$ is efficiently verifiable.
- $\mathsf{Encrypt}(\mathsf{PK}, m)$ To encrypt a message $m \in \mathbb{G}$, the algorithm generates a one-time signature key pair $(\mathsf{SSK}, \mathsf{SVK}) \leftarrow \mathcal{G}(1^\lambda)$, chooses $r, s \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ and computes the ciphertext as

$$C = (\mathsf{SVK}, C_1, C_2, D_1, D_2, E, \sigma) = \left(\mathsf{SVK},\ X^r,\ Y^s,\ (g^{\mathsf{SVK}} U)^r,\ (g^{\mathsf{SVK}} V)^s,\ m \cdot g^{r+s}, \sigma\right) ,$$

where $\sigma = \mathcal{S}(\mathsf{SSK}, (C_1, C_2, D_1, D_2))$.

– $\mathsf{ShareDecrypt}(\mathsf{PK}, i, \mathsf{SK}_i, C)$ Given $C = (\mathsf{SVK}, C_1, C_2, D_1, D_2, E, \sigma)$ and the private key share $\mathsf{SK}_i = (x_i, y_i)$, the algorithm first checks whether $\mathcal{V}(\mathsf{SVK}, \sigma, (C_1, C_2, D_1, D_2)) = 1$, $e(C_1, g^{\mathsf{SVK}}U) = e(X, D_1)$ and $e(C_2, g^{\mathsf{SVK}}V) = e(Y, D_2)$. If these tests are not all satisfied, it returns $(i, \perp)$. Otherwise, it computes $\hat{\mu}_i = (C_1^{x_i}, C_2^{y_i})$ and returns $\mu_i = (i, \hat{\mu}_i)$.

– $\mathsf{ShareVerify}(\mathsf{PK}, \mathsf{VK}_i, C, \mu_i)$ On input of PK, $\mathsf{VK}_i = (v_i, w_i)$, $C = (\mathsf{SVK}, C_1, C_2, D_1, D_2, E, \sigma)$ and $\mu_i = (i, \hat{\mu}_i)$, where $\hat{\mu}_i = (K_{i,1}, K_{i,2})$, this algorithm performs the following tests:

  Test 1: checks the validity of C as $\mathsf{ShareDecrypt}$ does.
  Test 2: checks whether $e(K_{i,1}, g) = e(C_1, v_i)$ and $e(K_{i,2}, g) = e(C_2, w_i)$.

  If these tests are both correct, it returns 1. If Test 1 fails, it outputs 1 iff $\hat{\mu}_i = \perp$. In any other case, it outputs 0.

– $\mathsf{Combine}(\mathsf{PK}, \mathbf{VK}, C, S)$ Parse S as $\{\mu_1, \ldots, \mu_t\}$ and μ_i as $(i, \hat{\mu}_i)$ for each i. If $\hat{\mu}_i = \perp$ for $i = 1, \ldots, t$, return $\perp$ (which indicates that C is invalid). Otherwise, parse $\hat{\mu}_i$ as $(K_{i,1}, K_{i,2})$ for each i and compute $K = \prod_{j \in S}(K_{i,1} \cdot K_{i,2})^{\Lambda_{i,S}}$ using Lagrange coefficients and finally obtain $m = E/K$.

The above description makes use of a one-time signature as in the original CHK transform [**?**]. We note that shorter ciphertexts can be obtained using a Waters-like hash function as in the repaired scheme of [**?**].

```
Benoit: Wouldn't it be better to describe the applications of PKENO
in the introduction?
```

Theorem 6. *The above TPKE scheme satisfies strong decryption consistency.*

Proof. The proof is quite similar to the proof of strong committingness for the repaired scheme of [**?**], which is given in Appendix B. It almost immediately follows from the verifiability properties of the bilinear map, which hold unconditionally. □

5 Conclusion

No conclusion for the time being.

A On the Notion of Proof Soundness

```
Georg: For the sake of consistent notation, I propose:
"(strong) proof soundness"
"a (strongly) proof-sound scheme" (with the dash)
"the scheme is (strongly) proof sound"
Any objections? Btw, I corrected some things in App. A (in particular
the statement of Proposition 2 should be correct now).
```

```
David: Agree
```

In this section, we compare the different notions of proof of soundness. In particular, we show that strong proof soundness is strictly stronger than the original notion of proof soundness, and that strong proof soundness and the committing property are incomparable.

Moreover, we compare the different notions of soundness in the KOSK model, where the adversary has to prove knowledge of the secret key. Within this model, we can show that the committing property is strictly stronger than strong soundness. We note that all our proofs also preserve IND-CCPA security in the sense that, if the strongly proof sound scheme has this property, then it is also stand-alone sound and still has this property. As for the separation we show that, if there exists a strongly proof sound (strongly committing, resp.) scheme which is also IND-CCPA, then there is an IND-CCPA scheme which is *not* strongly committing (strongly proof sound, resp.) but still proof sound. It is also easy to see that the case of adversarially chosen keys is strictly stronger, independently of the question whether the PKENO scheme is IND-CCPA secure or not.

A.1 Soundness vs. Strong Soundness

We show with the following proposition that strong proof soundness is strictly stronger than the notion of stand-alone soundness.

Proposition 1. *Any strongly proof sound PKENO scheme is also proof sound. Vice versa, there exists a proof sound PKENO scheme which is not strongly proof sound (assuming that there exists a proof sound scheme at all).*

Strong proof soundness implies the original soundness notion by letting the adversary run the key generation algorithm and output the derived public key. As for the converse, take any PKENO scheme and modify it such that the key generation algorithm appends a redundant bit 0 to all public keys (such that this bit is ignored by the encryption algorithm). The verification algorithm, too, ignores this bit —unless it is 1 in which case Ver accepts any input. Then, for honestly generated keys (ending with a 0-bit) the modified scheme obviously inherits stand-alone soundness, whereas the adversary can easily break our strong notion of soundness by outputting a public key with a final 1-bit. Additionally, the modified scheme preserves CCPA-security, correctness and completeness, because these properties are based on genuine public keys with a 0-bit.

A.2 Soundness vs. Committing Property

The next proposition proves that the committing property is strictly stronger than the notion of soundness.

Proposition 2. *Any strongly committing PKENO scheme is also proof sound. Vice versa, if a proof sound PKENO exists then there exists a scheme which is proof sound but not strongly committing.*

We omit the proof of this theorem, because it is analogous to the proof of Proposition 1.

A.3 Strong Soundness vs. Committing Property

The following proposition shows that strong proof soundness and the committing property are incomparable. As in the previous proposition, the separations preserve the IND-CCPA property.

Proposition 3. *If there exists a PKENO scheme which is strongly proof sound, then there exists a strongly proof sound PKENO scheme which is* not *strongly committing, and vice versa.*

Proof. Assume we are given a PKENO scheme which is strongly proof sound. Modify the scheme such that the encryption algorithm appends a redundant bit 0 to each ciphertext. The new decryption algorithms ignores the appended 0-bit but rejects ciphertexts ending with '1'. The new proof algorithm rejects all ciphertexts ending with a 1-bit and else runs the original proof algorithm with the 0-bit omitted. The modified verification algorithm now accepts any input (pk, C, m, π) where the ciphertext C ends with a 1-bit and otherwise operates as the original verification algorithm (for the ciphertext with the final 0-bit chopped off).

The modified scheme satisfies strong proof soundness if the original scheme does. This follows as the honestly generated ciphertext C in the new scheme always carries a '0' at the end and the verifier thus works as in the original scheme. In contrast, for the committing property, the adversary can easily generate ciphertexts C with a final 1-bit, making the verifier in the new scheme accept any message-proof pair. Furthermore, since the modified proof and decryption algorithms reject any malformed ciphertext with '1' and otherwise work as the original counterparts, CCPA-security is preserved. So are correctness and completeness by construction.

For the converse, assume we are given a strongly committing PKENO scheme. Change the scheme as follows: the key generation algorithm now appends a redundant 0-bit to each public key which the encryption algorithm ignores. The decryption algorithm and the proof algorithm work as before. Only the verifier now immediately accepts any input (pk, C, m, π) where pk ends with a '1' and where $m = pk$ (independently of C and π). For pk ending with '1' and $m \neq pk$ the verifier rejects. For well-formed public keys (with a 0-bit) the verifier works as before.

The modified scheme is clearly not strongly proof sound. An adversary picks two distinct messages m, m' where m' ends with '1', and outputs m and $pk = m'$. Then, after receiving C it outputs m' and an empty proof π'. By construction the verifier accepts (pk, C, m', π') as valid. Note also that the modified scheme preserves CCPA-security, correctness and completeness (as the key generation algorithm never returns malformed public keys). Finally, note that the scheme is still strongly committing because for public keys with a 1-bit appended the adversary can only make the verifier accept a single message, and for well-formed public keys this follows from the committing property of the original scheme. □

A.4 Strong Soundness vs. Committing Property in the KOSK Model

In the KOSK model [], however, where the adversary is required to additionally prove knowledge of a valid secret key, we can show that the committing property is strictly stronger. For simplicity, we model the KOSK by letting the adversary output the secret key as well. Note that this is not a restriction because we could also extract the key using rewinding techniques, or in the case that this is not possible, rely on the NIZKs due to Groth and Sahai [**?**]. More formally, in the KOSK model we assume that the adversary (in either soundness notion) in addition to pk also outputs sk such that sk, pk satisfy the correctness and completeness requirements of PKENO schemes. We then have:

Proposition 4. *In the KOSK model any strongly committing PKENO scheme is also strongly proof sound (but not vice versa).*

Proof. Assume we have an adversary $\mathcal{A}$ breaking the strong proof soundness. Then we construct an adversary $\mathcal{B}$ against the committing property as follows. Adversary $\mathcal{B}$ runs $\mathcal{A}$ to get (pk, sk, m), then computes $C \leftarrow \mathsf{Enc}(pk, m)$ and hands the ciphertext to $\mathcal{A}$ who replies with

(m', π'). Adversary $\mathcal{B}$ next computes $\pi \leftarrow \mathsf{Prove}(sk, m)$ and outputs $(pk, sk, C, m, \pi, m', \pi')$. By this $\mathcal{B}$ perfectly simulates $\mathcal{A}$'s game and thus succeeds with the same probability in the attack against the committing property as $\mathcal{A}$ does in the attack against proof soundness (using the fact that the secret key sk chosen by $\mathcal{A}$ obeys correctness and completeness).

The fact that the converse does not hold follows again from the separating example in Proposition 3 which still works in the KOSK model. □

B Repaired scheme [?]

Let $\mathbb{P} = (\mathbb{G}_1, \mathbb{G}_T, p, e)$ the description of a bilinear group. Let $\mathsf{H} : \{0,1\}^n \to \mathbb{G}_1$ be the hash function described as follows. On input of an integer n polynomially-bounded in k, the randomized hash key generator chooses $n+1$ random groups elements $h_0, \ldots, h_n \in \mathbb{G}_1$ and returns $h = (h_0, h_1, \ldots, h_n)$ as the hash function key. The hash function $\mathsf{H} : \{0,1\}^n \to \mathbb{G}_1^*$ is evaluated on a n-bit string $\boldsymbol{t} = (t_1, \ldots, t_n) \in \{0,1\}^n$ as the product $\mathsf{H}(\boldsymbol{t}) = h_0 \prod_{i=1}^{n} h_i^{t_i}$. In addition the scheme uses a collision-resistant hash function $\mathsf{CR} : \mathbb{G}_1 \times \{0,1\}^l \to \{0,1\}^n$. Let (E, D) be a a symmetric encryption scheme with keys' space $\mathbb{G}_T$. Galindo's PKENO scheme [**?**] is described in Figure 1.

Proof (of Theorem 1). We prove both claims of the theorem:
Commitment Property. Assume there is an adversary outputting $(pk, C, m, \pi, m', \pi')$ such that $\mathsf{Ver}(pk, C, m, \pi) = 1 = \mathsf{Ver}(pk, C, m', \pi')$ and $m \neq m'$. We parse the output as $pk = (1^k, \mathbb{P}, \mathsf{E}, \mathsf{D}, \mathsf{CR}, h, Y)$, $C = (c_0, c_1, c_2) \in \mathbb{G}^3$. Define $\boldsymbol{t} := \mathsf{CR}(c_0, c_1)$. We distinguish two cases:

1. $e(g, c_2) \neq e(H(\boldsymbol{t}), c_1)$
   Since in this case, Ver only outputs 1 if $m = \perp$, this case cannot occur since $m \neq m'$.
2. $e(g, c_2) = e(H(\boldsymbol{t}), c_1)$
   First, this means that there exists r s.t. $c_1 = g^r$ and $c_2 = H(\boldsymbol{t})^r$. Second, in this case, Ver only outputs 1 if $\pi \neq \emptyset$, we parse thus π as (d_1, d_2) and π' as (d_1', d_2'). Moreover, since both proofs pass verification, we have $e(g, d_1) = Y \cdot e(H(\boldsymbol{t}), d_1)$ and $e(g, d_1') = Y \cdot e(H(\boldsymbol{t}), d_1')$. Letting $Y = e(g, g)^\alpha$ for some unknown α, there exist s and s' such that $d_2 = g^s$, $d_1 = g^\alpha H(\boldsymbol{t})^s$, $d_2' = g^{s'}$, $d_1' = g^\alpha H(\boldsymbol{t})^{s'}$.
   In the verification of π, the key K is computed as

$$K = e(c_1, d_1)/e(c_2, d_2) = e(g^r, g^\alpha H(\boldsymbol{t})^s)/e(H(\boldsymbol{t})^r, g^s) = e(g^r, g^\alpha) \ .$$

   For π', this computation analogously gives $K' = e(g^r, g^\alpha) = K$, which means $\mathsf{D}(K, c_0)$ yields the same candidate message for both π and π'. Since $m \neq m'$, at most one of the verifications returns 1, which is a contradiction.

Strong Proof Soundness. Suppose there is an adversary $\mathcal{A}$ that wins the game in Definition 3. Let $pk = (1^k, \mathbb{P}, \mathsf{E}, \mathsf{D}, \mathsf{CR}, h, Y)$ and $m \in \mathcal{M}_{pk}$ be its output in the first stage. Choose $r \leftarrow \mathbb{Z}_p$ and set $c_0 = \mathsf{E}(Y^r, m)$, $c_1 = g^r$, $c_2 = H(\boldsymbol{t})^r$ with $\boldsymbol{t} = \mathsf{CR}(c_0, c_1)$ and send (c_0, c_1, c_2) to $\mathcal{A}$.

Suppose $\mathcal{A}$ outputs (m', π') such that $\mathsf{Ver}(pk, C, m', \pi') = 1$ and $m \neq m'$. Since we have $e(g, c_2) = e(H(\boldsymbol{t}), c_1)$, m' must be different from $\perp$—otherwise Ver outputs 0. Moreover, $\pi' = (d_1', d_2')$ must satisfy $e(g, d_1') = Y \cdot e(H(\boldsymbol{t}), d_2')$, which means that there exists s' such that $d_2' = g^{s'}$, $d_1' = g^\alpha H(\boldsymbol{t})^{s'}$, with α such that $Y = e(g, g)^\alpha$. As in the first part of the

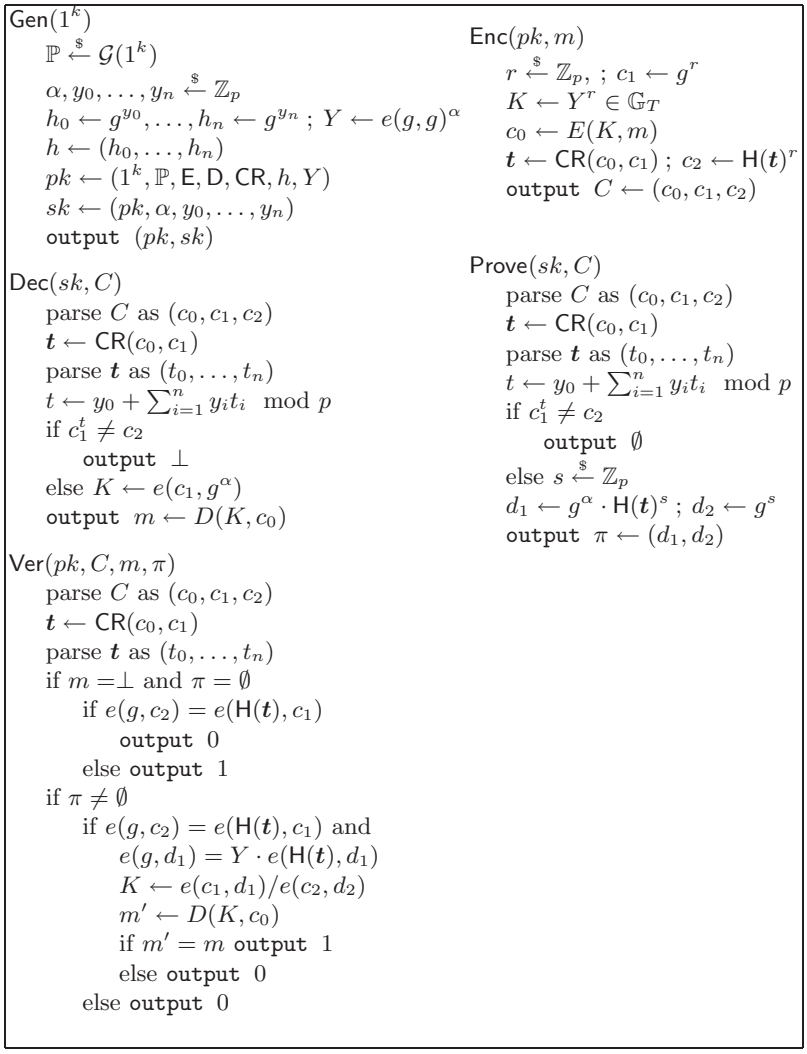

Figure 1. Galindo scheme.

proof, the key K' produced by Ver is $e(g^r, g^\alpha)$, i.e., $K' = Y^r$. By correctness of the symmetric encryption, the candidate message in Ver is m, the message output in the first stage. This contradicts the assumption that Ver outputs 1 on $m' \neq m$. □