# Strong Knowledge Extractors for Public-Key Encryption Schemes

M. Barbosa[1] and P. Farshim[2]

[1] CCTC/Departamento de Informática, Universidade do Minho,
Campus de Gualtar, 4710-057 Braga, Portugal.
mbb@di.uminho.pt
[2] Information Security Group, Royal Holloway, University of London,
Egham, Surrey, TW20 0EX, United Kingdom.
Pooya.Farshim@rhul.ac.uk

**Abstract.** Completely non-malleable encryption schemes resist attacks which allow an adversary to tamper with both ciphertexts *and* public keys. In this paper we introduce two extractor-based properties that allow us to gain insight into the design of such schemes and to go beyond known feasibility results in this area. We formalise *strong plaintext awareness* and *secret key awareness* and prove their suitability in realising these goals. Strong plaintext awareness imposes that it is infeasible to construct a ciphertext under *any* public key without knowing the underlying message. Secret key awareness requires it to be infeasible to produce a new public key without knowing a corresponding secret key.
We study the relations among these and existing notions in the literature and show that if such properties are realisable (and one admits non-black-box simulators) then the impossibility result established for the construction of completely non-malleable schemes under non-assisted simulators no longer holds. We also look at how such notions can be realised in the standard model and in the random oracle model. More precisely, we propose a generic transformation to construct secret key aware schemes in the random oracle model and give preliminary steps towards building such schemes in the standard model. To this end, we introduce a novel factorisation-based knowledge assumption, which roughly speaking, requires it to be infeasible to construct integers of the form $P^2Q$ without knowing the corresponding factorisation.

**Keywords.** Secret Key Awareness. Strong Plaintext Awareness. Knowledge of Factorisation. Strong Chosen-Ciphertext Attacks. Complete Non-Malleability. Public-Key Encryption.

## 1 Introduction

BACKGROUND. Indistinguishability of ciphertexts under chosen-ciphertext attacks (IND-CCA2) is a convenient reformulation of a more intuitive security notion known as *non-malleability*. Roughly speaking, an encryption scheme is non-malleable if, given a challenge ciphertext, it is infeasible to output a new ciphertext encrypting a plaintext related in a "meaningful" or "interesting" way to that enclosed in the challenge. The advantages of the indistinguishability formulation become apparent when one considers various subtleties which arise when defining what a meaningful relation is [PSV07,BS06]. Recently, Fischlin [Fis05] has considered the problem of using public key encryption schemes to build non-malleable commitment schemes. It has been shown that the standard definition of non-malleability is not sufficient for this application and that a stronger variant, referred to as *complete non-malleability*, is required. This security definition allows the adversary to maul the challenge public key, as well as the ciphertext. Put differently, the adversary can output a related ciphertext under a new public key of its choice. Unlike standard non-malleability, it has been shown in [Fis05] that completely non-malleable schemes are hard to construct. In particular, such schemes do not exist for general relations with respect to black-box simulators that cannot access a decryption oracle (i.e. non-assisted simulators).

Complete non-malleability has recently been shown to be equivalent to *indistinguishability under strong chosen-ciphertext attacks* [BF10,ARP03]. This model enhances the adversary's capabilities to forge public keys and ask the decryption oracle to provide decryptions under the corresponding (possibly unknown) secret keys. It was also shown that it is possible to construct efficient completely non-malleable schemes using the strong chosen-ciphertext attack model, which is more convenient than performing the proof in the original simulation-based definition. Unfortunately, the equivalence result connecting strong chosen-ciphertext security to complete non-malleability holds

only for simulators *assisted* by a strong decryption oracle. It therefore remains an open problem to construct efficient schemes that achieve complete non-malleability in the strongest sense.

The impossibility result from [Fis05] dictates that to construct a scheme that achieves complete non-malleability with respect to *non-assisted* simulators, one must resort to a non-black-box simulator. In this paper we consider extractor-based properties that allow us to gain insight into the design of completely non-malleable schemes and provide a technique to go beyond known feasibility results in this area. We formalise *strong plaintext awareness* and *secret key awareness* and prove their suitability in realising these goals. We show that if such properties are realisable, and one considers non-black-box simulators, then the impossibility result established for non-assisted simulators no longer holds. We also look at how such notions can be realised with and without random oracles.

STRONG PLAINTEXT AWARENESS. Plaintext awareness formalises the intuition that one can achieve security under chosen-ciphertext attacks by making it infeasible to construct a valid ciphertext without *knowing*, a priori, the message hidden inside it. In fact, it has been shown that the combination of plaintext awareness and semantic security is enough to achieve chosen-ciphertext security [BP04]. We formulate a natural strengthening of plaintext awareness that requires the existence of a strong plaintext extractor that decrypts ciphertexts, even if they are encrypted under adversarially generated public keys. We prove a fundamental theorem according to which a *strongly plaintext-aware* (SPA) and IND-CPA secure scheme also withstands strong chosen-ciphertext attacks[3]. This implies, through the results in [BF10], that such a scheme is also completely non-malleable with respect to assisted simulators. We extend this result by showing that strong plaintext awareness allows us to directly build *non-assisted* simulators. The resulting simulators depend on the adversary and hence they are not black-box. This permits going around the impossibility result established by Fischlin [Fis05]. Furthermore, strong plaintext awareness generalises a proof technique used by Fischlin to demonstrate that (a slightly modified version of) RSA-OAEP is completely non-malleable for non-assisted simulators[4].

SECRET KEY AWARENESS. We also propose a new extractor-based security definition that takes a different perspective on how to achieve strong plaintext awareness and complete non-malleability. Roughly speaking, this notion that we call *secret key awareness* (SKA), requires it to be infeasible to generate new valid public keys without knowing their corresponding secret keys. It therefore looks at enhancing the security of key-generation mechanisms. We show that an encryption scheme that is secret key aware and IND-CCA2 is also secure under strong chosen-ciphertext attacks, and therefore completely non-malleable. We derive this result via a stronger indistinguishability security notion, where the adversary has access to a public key inversion oracle[5]. Furthermore, we prove that secret key awareness, together with standard plaintext awareness, implies strong plaintext awareness. Hence, secret key awareness provides all of the benefits of strong plaintext awareness. Additionally, secret key awareness permits the construction of a complete non-malleability simulator that *opens* the secret key associated with the public created by the adversary. This is particularly relevant when the scheme is used in commitment schemes, where to de-commit one reveals a secret key rather a message/randomness pair. Strong plaintext awareness in not sufficient to open a ciphertext in the sense of de-commitment, as it does not guarantee knowledge of the *randomness* used in encryption.

SCHEMES. We propose a generic transformation that permits transforming any IND-CCA2 scheme into a secret key aware (and still IND-CCA2) scheme in the random oracle model. The resulting schemes are therefore completely non-malleable for non-assisted simulators. We also take first steps towards building fully secret-key-aware schemes without random oracles. We propose a *generic* construction inspired in escrow public-key encryption [BNB07], relying on schemes whose key-generation routines themselves operate as an encryption scheme. We are, however, unable to instantiate this scheme and leave it as an interesting open problem. Next, we move to specific construc-

[3] This result also has applications in certificateless encryption [ARP03].

[4] A corollary of this result is that we obtain a new perspective on the *standard* notion of plaintext awareness. Indeed, a similar proof strategy can be used to construct non-assisted simulators for standard non-malleability.

[5] This type of oracle has been shown to have numerous applications in the context of adaptive one-way functions [PPV08].

tions based on knowledge assumptions. A natural candidate for building a secret key aware scheme is the Diffie-Hellman Knowledge assumption [BP04]. This approach, however, fails once we notice that secret key awareness allows adaptive attacks on the public keys, whereas Diffie-Hellman tuples are malleable. We therefore introduce a new knowledge assumption stating, roughly speaking, that it is impossible to compute integers of the form $P^2Q$, where $P$ and $Q$ are prime, without knowing the factors and even if provided with another integer of this form. This assumption can be used to demonstrate that variants of RSA satisfy weak forms of secret key awareness.

ORGANISATION. We first review related work. Then, in Section 2 we settle notation and recall the syntax for public-key encryption schemes. We also recall the definition of strong decryption oracles and IND-SCCA2 security. In the same section we also introduce invert and chosen-ciphertext attacks, which we will use later on in the paper. In Section 3 we introduce our extractor-based notions. In Section 4 we discuss constructions of secret key aware schemes.

### 1.1 Related work

Plaintext awareness was originally formulated by Bellare and Rogaway [BR94] in the random oracle model as a stepping stone towards establishing the security of RSA-OAEP. It heavily relied on the extractability property of random oracles to reverse engineer the ciphertexts created by the adversary and recover the plaintext within. This technique was shown to be sufficient to make IND-CPA schemes resilient to chosen-ciphertext attacks. Plaintext awareness was later adapted to the standard model. The first work in this area was that of Herzog, Liskov, and Micali [HLM03], who defined plaintext awareness in a scenario where the *encryptor* is also registered with a Certification Authority. Later, Bellare and Palacio [BP04] gave the first definition of PA for the standard scenario where only the receiver belongs to a public-key infrastructure. It was also shown that the Damgård-ElGamal and Cramer-Shoup-*lite* encryption schemes satisfy PA1, a non-adaptive flavour of plaintext awareness where the adversary does not have access to ciphertexts it has not created itself. Later, Dent [Den06] showed that the Cramer-Shoup encryption scheme is fully plaintext aware (PA2), solving a problem left open by Bellare and Palacio. This proof was recently simplified by Teranishi and Ogata [TO08a]. Di Raimondo, Gennaro, and Krawczyk [RGK06] give a key-exchange protocol where the full power of PA (and not just IND-CCA2 security) is needed, thereby providing evidence that PA may be of interest beyond its application to CCA security.

The relation between plaintext awareness and other security notions for public-key encryption schemes has been extensively studied. It is well known [BP04,BDPR98] that plaintext awareness together with IND-CPA imply a level of security that is strictly stronger than IND-CCA2. Fujisaki [Fuj06] further explored this gap in the random oracle model and introduced a notion of plaintext simulatability which suffices to achieve IND-CCA2 but is weaker than plaintext awareness. The authors in [TO08c] investigate how the weaker notion of PA0, where only a single non-adaptively generated ciphertext must be extracted, can be altered so that it implies the strongest flavours of plaintext awareness where multiple (adaptive) decryption queries must be answered. However, this requires modifying the standard definitions extending adversaries and extractors so that they take auxiliary input information and have access to a decryption oracle. The same authors further show in [TO08b] that plaintext awareness is an "all-or-nothing property" in the sense that *one-wayness* (or even a weaker condition called non-triviality) together with PA2 plaintext awareness is enough to guarantee IND-CCA2 security. Birkett and Dent [BD08] settled the relations between various notion of plaintext awareness introduced in [Den06], and using the "channel" technique of [TO08b] showed that schemes with infinite message spaces that are plaintext-aware and one-way do not exist.

Non-malleability (as a general notion) was originally introduced in the seminal work of Dolev, Dwork, and Naor [DDN00,DDN91]. In order to establish relations with other notions of security, non-malleability for public-key encryption was reformulated by Bellare et al. [BDPR98] as a comparison-based security model. Bellare and Sahai [BS06,BS99] later fully established the relation between this comparison-based definition and the original simulation-based definition of Dolev et al. Pass, Shelat, and Vaikuntanathan [PSV07] provide a full characterisation of non-malleability, identifying some shortcomings in previous results and considering their robustness under a form of composition where the adversary is provided with a polynomial number of challenge ciphertexts. It has

also been shown that, for general schemes, simulation-based non-malleability is strictly stronger than comparison-based non-malleability.

Complete non-malleability, was proposed by Fischlin [Fis05]. Here the adversary is allowed to choose the public key under which the target ciphertext is produced. The same author presented impossibility results as to the construction of completely non-malleable schemes with respect to black-box simulators and general relations, and showed that a modified version of RSA-OAEP is completely non-malleable in the random oracle model. Visconti and Ventre [VV08] proposed a comparison-based definition of complete non-malleability, studied its relation with the simulation-based definition of Fischlin, and also gave a generic construction of completely non-malleable schemes based on non-interactive zero-knowledge proofs-of-knowledge. The authors in [BF10] define strong decryption oracles, use this to introduce indistinguishability under strong chosen-ciphertext attacks and establish relations with assisted simulation-based and comparison-based complete non-malleability. A practical and strongly secure scheme (without random oracles) based on the decisional bilinear Diffie-Hellman problem is also given.

Adaptive one-way functions [PPV08], where an adversary has access to an inversion oracle, and extractable one-way functions [CD08,CD09], where one requires knowledge of pre-image, have been recently proposed. Secret key awareness can be seen as a refinement of these notions for public-key encryption.

## 2 Preliminaries

NOTATION. We write $x \leftarrow y$ for assigning value $y$ to variable $x$, and $x \leftarrow_{\$} X$ for sampling $x$ from set $X$ uniformly at random. If $X$ is empty, we set $x \leftarrow \perp$, where $\perp \notin \{0,1\}^{\star}$ is a special failure symbol. If $A$ is a probabilistic algorithm we write $x \leftarrow_{\$} A(I_1, I_2, \ldots)$ for the action of running $A$ on inputs $I_1, I_2, \ldots$ with random coins chosen uniformly at random, and assigning the result to $x$. When $A$ is run on specific coins $r$, we write $x \leftarrow A(I_1, I_2, \ldots; r)$. We denote boolean values, namely the output of checking whether a relation holds, by $\mathsf{T}$ (true) and $\mathsf{F}$ (false). For a space $\mathsf{Sp} \subseteq \{0,1\}^{\star}$, we identify $\mathsf{Sp}$ with its characteristic function. In other words, $\mathsf{Sp}(s) = \mathsf{T}$ if and only if $s \in \mathsf{Sp}$. The function $\mathsf{Sp}(\cdot)$ always exists, although it may not be computable in polynomial time. We say $s$ is valid with respect to $\mathsf{Sp}$ if and only if $\mathsf{Sp}(s) = \mathsf{T}$. When this is clear from the context, we also use $\mathsf{Sp}$ for sampling uniformly from $\mathsf{Sp}$. Unless stated otherwise, the range of a variable $s$ is assumed to be $\{0,1\}^{\star}$. The symbol : is used for appending an element to a list, and we indicate vectors using bold-faced font. We say $f(\lambda)$ is negligible if $f(\lambda) \in \cap_{c \in \mathbb{N}} O(\lambda^{-c})$.

GAMES. In this paper we will be using code-based game-playing [BR06]. Each game has an **Initialize** and a **Finalize** procedure. It also has specifications of procedures to respond to an adversary's various oracle queries. A game $\mathsf{Game}$ is run with an adversary $\mathcal{A}$ as follows. First **Initialize** runs and its outputs are passed to $\mathcal{A}$. Then $\mathcal{A}$ runs and its oracle queries are answered by the procedures of $\mathsf{Game}$. These procedures return $\perp$ if queried on $\perp$. When $\mathcal{A}$ terminates, its output is passed to **Finalize** which returns the outcome of the game $y$. This interaction is written as $\mathsf{Game}^{\mathcal{A}} \Rightarrow y$. In each game, we restrict attention to *legitimate* adversaries. Legitimacy is defined specifically for each game. All algorithms (adversaries, extractors and plaintext/public-key creators) are assumed to run in probabilistic polynomial time (PPT).

PUBLIC-KEY ENCRYPTION. We adopt the standard multi-user syntax with the extra $\mathsf{Setup}$ algorithm [BBM00], which we believe is the most natural one for security models involving multiple public keys. A public-key encryption scheme $\Pi = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{MsgSp}, \mathsf{Enc}, \mathsf{Dec})$ is specified by five polynomial-time algorithms (in the length of their inputs) as follows. $\mathsf{Setup}$ is the probabilistic setup algorithm which takes as input the security parameter $1^{\lambda}$ and returns the common domain parameter[6] $\mathsf{I}$. $\mathsf{Gen}(\mathsf{I})$ is the probabilistic key-generation algorithm. On input global parameters $\mathsf{I}$, this algorithm returns a secret key $\mathsf{SK}$ and a matching public key $\mathsf{PK}$. Algorithm $\mathsf{MsgSp}(\mathsf{m}, \mathsf{PK})$ is a deterministic message space recognition algorithm. On input $\mathsf{m}$ and $\mathsf{PK}$ this algorithm returns $\mathsf{T}$ or $\mathsf{F}$. $\mathsf{Enc}(\mathsf{m}, \mathsf{PK}; r)$

[6] Although all algorithms are parameterised by $\mathsf{I}$, we often omit $\mathsf{I}$ as an explicit input for readability. Furthermore, we assume that the security parameter is included in $\mathsf{I}$.

is the probabilistic encryption algorithm. On input a message $\mathsf{m}$, a public key $\mathsf{PK}$, and possibly some random coins $r$, this algorithm outputs a ciphertext $\mathsf{c}$ or a special failure symbol $\perp$. Finally, $\mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK})$ is the deterministic decryption algorithm. On input of a ciphertext $\mathsf{c}$ and keys $\mathsf{SK}$ and $\mathsf{PK}$, this algorithm outputs a message $\mathsf{m}$ or a special failure symbol $\perp$. The correctness of a public-key encryption scheme requires that for any $\mathsf{I} \leftarrow_{\$} \mathsf{Setup}(1^{\lambda})$, any $(\mathsf{SK}, \mathsf{PK}) \leftarrow_{\$} \mathsf{Gen}(\mathsf{I})$ and all $\mathsf{m} \in \mathsf{MsgSp}(\mathsf{PK})$ we have $\Pr[\mathsf{Dec}(\mathsf{Enc}(\mathsf{m}, \mathsf{PK}), \mathsf{SK}, \mathsf{PK}) = \mathsf{m}] = 1$.

REMARK. We note that the multi-user syntax permits capturing in the same framework schemes that execute in the standard model, in which case the global parameters are empty; and also schemes which execute in the Common Reference String (CRS) model. *All the relations that we establish between the different models apply to both cases*.

VALIDITY CHECKING ALGORITHMS. The following spaces (and associated functions) will be used throughout the paper. All of these spaces are parameterised by $\mathsf{I}$ and are subsets of $\{0,1\}^{\star}$.

$$\mathsf{MsgSp}(\mathsf{PK}) := \{\mathsf{m} : \mathsf{MsgSp}(\mathsf{m}, \mathsf{PK})\}$$
$$\mathsf{KeySp} := \{(\mathsf{SK}, \mathsf{PK}) : \exists r\ (\mathsf{SK}, \mathsf{PK}) = \mathsf{Gen}(r)\}$$

We assume throughout the paper that the encryption and decryption algorithms check if $\mathsf{m} \in \mathsf{MsgSp}(\mathsf{PK})$ and return $\perp$ if it does not hold. Often the algorithm $\mathsf{MsgSp}$ does not depend on $\mathsf{PK}$ in the sense that for any two valid public keys $\mathsf{PK}$ and $\mathsf{PK}'$ and any $\mathsf{m} \in \{0,1\}^{\star}$ we have $\mathsf{MsgSp}(\mathsf{m}, \mathsf{PK}) = \mathsf{MsgSp}(\mathsf{m}, \mathsf{PK}')$. For general schemes, one can consider the infinite message space $\mathsf{MsgSp}(\mathsf{PK}) = \{0,1\}^{\star}$ case. However, given that in this paper we will often consider the set of all valid messages and sample from it, we restrict our attention to schemes with finite message spaces. As pointed out by Pass et al. [PSV07], this means that to avoid degenerate cases we must also restrict our attention to schemes for which all the elements in the range of decryption can be efficiently encrypted[7], including the special failure symbol $\perp$. We also assume that the key-pair validity algorithm $\mathsf{KeySp}$ is polynomial-time and require that decryption returns $\perp$ if this check fails on the key-pair passed to it. We also assume various algorithms check for structural properties such as correct encoding, membership in a group, etc.

### 2.1 Strong chosen-ciphertext security

The idea behind a strong chosen-ciphertext attack is to give the adversary access to an oracle that decrypts ciphertexts of the adversary's choice with respect to arbitrary public keys.

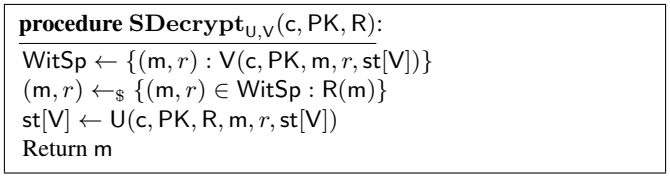


Fig. 1: Generic definition of a strong decryption oracle.

We follow [BF10] and adopt a generic definition of strong decryption as shown in Figure 1. The oracle proceeds in three steps. The first step models the general procedure of constructing a set of candidate (valid) decryption results[8]. The second step consists of choosing one of these candidate solutions to return to the adversary. The final step updates the state of the oracle, if it keeps one[9]. As discussed in [BF10] the motivation for having such a

[7] This can be easily achieved for schemes used in practice.

[8] Search for messages is over sufficiently long bit strings together with the special symbol $\perp$. Search for random coins is over sufficiently long bit strings.

[9] The state is initialized to some value $\mathsf{st}_0$. A natural use of the state is to make sure that decryption is consistent in different calls.

general definition is that the notion of *the message encapsulated by the ciphertext* can be defined in a number of ways, depending on the witnesses that are taken to assess the validity of the public key/ciphertext pair. For example, one can define validity via the encryption operation, in which case a message/randomness pair is the witness

$$\mathsf{V}(\mathsf{c}, \mathsf{PK}, \mathsf{m}, r) := \mathsf{c} \stackrel{?}{=} \mathsf{Enc}(\mathsf{m}, \mathsf{PK}; r), \tag{1}$$

or via the decryption algorithm, where a message/secret key pair is the witness

$$\mathsf{V}'(\mathsf{c}, \mathsf{PK}, \mathsf{m}, r) := (\mathsf{SK}, \mathsf{PK}) \stackrel{?}{=} \mathsf{Gen}(r) \wedge \mathsf{m} \stackrel{?}{=} \mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK}). \tag{2}$$

Note that neither criterion guarantees that, if a message is found to be a valid decryption result, then it will be unique. This is because the correctness restriction guarantees *unique decryptability* only for correctly constructed $(\mathsf{c}, \mathsf{PK})$ pairs: it says nothing about the result of decryption when an invalid public key and/or an invalid ciphertext are provided as inputs. This justifies the need for the second step in the definition we propose: there could be many valid decryption results to choose from, and it is left to the adversary to control how this is done.
Note that, for a well-defined and broad class of schemes [BF10], this general definition collapses into a much simpler one. However, we follow this approach for the sake of generality.

We now present the definition of ciphertext indistinguishability under strong chosen-ciphertext attacks, introduced in [BF10] as the natural extension of the standard notion of security for public-key encryption schemes. The IND-SCCAx advantage of an adversary $\mathcal{A}$ for $\mathsf{x} = 0, 1, 2$ against $\Pi$ is defined by

$$\mathbf{Adv}^{\mathsf{ind\text{-}sccax}}_{\Pi,\mathcal{A}}(\lambda) := 2 \cdot \Pr\left[\mathsf{IND\text{-}SCCAx}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T}\right] - 1,$$

where game IND-SCCAx is shown in Figure 2. Implicit in this definition are the descriptions of the U and V algorithms, which are fixed when analysing a scheme in the resulting IND-SCCAx model. We say a scheme is IND-SCCAx secure if the advantage of any PPT adversary is negligible.

Game $\mathsf{IND\text{-}SCCAx}_{\Pi}(\lambda)$

**procedure** $\mathbf{Initialize}(\lambda)$:
$b \leftarrow_{\$} \{0,1\}$; $\mathsf{I} \leftarrow_{\$} \mathsf{Setup}(1^{\lambda})$
$(\mathsf{SK}^{\star}, \mathsf{PK}^{\star}) \leftarrow_{\$} \mathsf{Gen}(\mathsf{I})$
$\mathsf{List} \leftarrow []$; $\mathsf{st}[\mathsf{V}] \leftarrow \mathsf{st}_0$
Return $(\mathsf{I}, \mathsf{PK}^{\star})$

**procedure** $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$:
$\mathsf{c} \leftarrow_{\$} \mathsf{Enc}(\mathsf{m}_b, \mathsf{PK}^{\star})$
$\mathsf{List} \leftarrow (\mathsf{c}, \mathsf{PK}^{\star}) : \mathsf{List}$
Return c

**procedure** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$:
Return $\mathbf{SDecrypt}_{\mathsf{U},\mathsf{V}}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$

**procedure** $\mathbf{Finalize}(b')$:
Return $(b' = b)$

Fig. 2: Game defining indistinguishability under strong chosen-ciphertext attacks. An adversary $\mathcal{A}$ is legitimate if: 1) It calls **Left-Right** only once with $\mathsf{m}_0, \mathsf{m}_1 \in \mathsf{MsgSp}(\mathsf{PK})$ such that $|\mathsf{m}_0| = |\mathsf{m}_1|$; and 2) R is polynomial-time and, if $\mathsf{x} = 0$ it does not call **SDecrypt**, if $\mathsf{x} = 1$ it does not call **SDecrypt** after calling **Left-Right**, and if $\mathsf{x} = 2$ it does not call **SDecrypt** with a tuple $(\mathsf{c}, \mathsf{PK})$ in List.

### 2.2 Security under invert and chosen-ciphertext attacks

We introduce a new security model for encryption that helps us clarify the relations among notions we establish later on. In this model, the adversary has access to an oracle that, given a public key generated by the adversary, provides it with the corresponding secret key. Figure 4 presents the general form of the **Invert** procedure, which is analogous to the **SDecrypt** procedure presented in the previous section. When many secret keys satisfy the validity criterion, the adversary is also allowed to restrict the set of "interesting" secret keys from which the answer is sampled by providing a relation R on secret keys as input to the oracle. A natural validity criteria for this oracle is

$$\mathsf{V}(\mathsf{PK}, \mathsf{SK}, r) := (\mathsf{SK}, \mathsf{PK}) \stackrel{?}{=} \mathsf{Gen}(r)$$

Game $\mathsf{IND\text{-}ICAx}_{\Pi}(\lambda)$

**procedure** $\mathbf{Initialize}(\lambda)$:
$b \leftarrow_{\$} \{0,1\}$; $\mathsf{I} \leftarrow_{\$} \mathsf{Setup}(1^{\lambda})$
$(\mathsf{SK}^{\star}, \mathsf{PK}^{\star}) \leftarrow_{\$} \mathsf{Gen}(\mathsf{I})$
$\mathsf{List} \leftarrow []$; $\mathsf{st}[\mathsf{V}] \leftarrow \mathsf{st}_0$
Return $(\mathsf{I}, \mathsf{PK}^{\star})$

**procedure** $\mathbf{Invert}(\mathsf{PK}, \mathsf{R})$:
Return $\mathbf{Invert}_{\mathsf{U},\mathsf{V}}(\mathsf{PK}, \mathsf{R})$

**procedure** $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$:
$\mathsf{c} \leftarrow_{\$} \mathsf{Enc}(\mathsf{m}_b, \mathsf{PK}^{\star})$
$\mathsf{List} \leftarrow (\mathsf{c}, \mathsf{PK}^{\star}) : \mathsf{List}$
Return $\mathsf{c}$

**procedure** $\mathbf{Decrypt}(\mathsf{c})$:
Return $\mathsf{Dec}(\mathsf{c}, \mathsf{SK}^{\star}, \mathsf{PK}^{\star})$

**procedure** $\mathbf{Finalize}(b')$:
Return $(b' = b)$

Fig. 3: Game defining indistinguishability under invert and chosen-ciphertext attacks. An adversary $\mathcal{A}$ is legitimate if: 1) It calls $\mathbf{Left\text{-}Right}$ only once with $\mathsf{m}_0, \mathsf{m}_1 \in \mathsf{MsgSp}(\mathsf{PK})$ such that $|\mathsf{m}_0| = |\mathsf{m}_1|$; 2) $\mathsf{R}$ is polynomial-time and, if $\mathsf{x} = 0$ it does not call $\mathbf{Decrypt}$ or $\mathbf{Invert}$, if $\mathsf{x} = 1$ it does not call $\mathbf{Decrypt}$ or $\mathbf{Invert}$ after calling $\mathbf{Left\text{-}Right}$, and if $\mathsf{x} = 2$ it does not call $\mathbf{Decrypt}$ with a $\mathsf{c}$ in $\mathsf{List}$; and 3) It does not call $\mathbf{Invert}$ on $\mathsf{PK}^{\star}$.

**procedure** $\mathbf{Invert}_{\mathsf{U},\mathsf{V}}(\mathsf{PK}, \mathsf{R})$:
$\mathsf{WitSp} \leftarrow \{(\mathsf{SK}, r) : \mathsf{V}(\mathsf{PK}, \mathsf{SK}, r, \mathsf{st}[\mathsf{V}])\}$
$(\mathsf{SK}, r) \leftarrow_{\$} \{(\mathsf{SK}, r) \in \mathsf{WitSp} : \mathsf{R}(\mathsf{SK})\}$
$\mathsf{st}[\mathsf{V}] \leftarrow \mathsf{U}(\mathsf{PK}, \mathsf{R}, \mathsf{SK}, r, \mathsf{st}[\mathsf{V}])$
Return $\mathsf{SK}$

Fig. 4: Generic definition of an invert oracle.

accepting all key-pairs that may be output by the key-generation algorithm[10].

We define the IND-ICAx advantage of an adversary $\mathcal{A}$ for $\mathsf{x} = 0, 1, 2$ against encryption scheme $\Pi$ is defined by

$$\mathbf{Adv}^{\mathsf{ind\text{-}icax}}_{\Pi,\mathcal{A}}(\lambda) := 2 \cdot \Pr\left[\mathsf{IND\text{-}ICAx}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T}\right] - 1,$$

where Game IND-ICAx is shown in Figure 3. We say a scheme is IND-ICAx secure if the advantage of any PPT adversary is negligible. This new model can be related to IND-SCCAx as follows. For a given $\mathbf{Invert}_{\mathsf{U},\mathsf{V}}$ procedure, define the associated $\mathbf{SDecrypt}_{\mathsf{U}',\mathsf{V}'}$ procedure through the algorithms shown in Figure 5. The following theorem shows that security under each possible definition of an invert oracle implies IND-SCCAx security under a well-defined version of the strong decryption oracle.

**Theorem 1** (IND-ICAx $\Rightarrow$ IND-SCCAx)**.** *Let $\mathcal{A}$ be an* IND-SCCAx *adversary against encryption scheme $\Pi$ with respect to $\mathbf{SDecrypt}_{\mathsf{U}',\mathsf{V}'}$ associated to $\mathbf{Invert}_{\mathsf{U},\mathsf{V}}$ as defined in Figure 5. Then there exists an* IND-ICAx *adversary $\mathcal{A}_1$ against $\Pi$ with at least the same advantage as that of $\mathcal{A}$.*

The reduction is constructed by simulating the strong decryption oracle using both the standard decryption oracle (for queries under the challenge public key) and the invert oracle (for adversarially chosen-keys) available in the IND-ICAx game. The details are given in Appendix A. The interesting part of the proof is an argument showing that this simulation fits into the generic structure of $\mathbf{SDecrypt}$ given in Figure 1 and, in particular, that the effect of the relation $\mathsf{R}$ passed to the strong decryption oracle can be emulated through a relation $\mathsf{R}'$ passed to the invert oracle. The intuition here is that the strong decryption oracle associated with a particular invert oracle maps the relation $\mathsf{R}$ that allows the adversary to restrict the set of interesting messages onto a relation $\mathsf{R}'$ that selects the set of secret keys that decrypt the queried ciphertext into the same set of interesting messages. Technically, the relation $\mathsf{R}'_{\mathsf{c},\mathsf{PK}}(\mathsf{SK}) := \mathsf{R}(\mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK}))$ allows us to simulate the oracle in Figure 5 with the correct distribution.

## 3 Extractor-based properties

The strong chosen-ciphertext security model that was recalled in Section 2 suggests that any secure scheme under this definition must ensure, by construction, that strong decryption queries are of no help to the adversary even when the associated public keys are chosen adaptively. Plaintext awareness [BP04] formalises this intuition when a

[10] Another validity criteria, corresponding to a natural stateful invert oracle, will ensure that repeat queries will be answered consistently.

**algorithm** $\mathsf{V}'(\mathsf{c}, \mathsf{PK}, \mathsf{m}, r, \mathsf{st}[\mathsf{V}'])$:
$(\mathsf{st}[\mathsf{V}], (\mathsf{SK}^*, \mathsf{PK}^*)) \leftarrow \mathsf{st}[\mathsf{V}']$
If $\mathsf{PK} = \mathsf{PK}^\star$ Then
  If $\mathsf{m} = \mathsf{Dec}(\mathsf{c}, \mathsf{SK}^\star, \mathsf{PK}^\star)$ Return $\mathsf{T}$ Else Return $\mathsf{F}$
$(\mathsf{SK}, r') \leftarrow r$
If $\mathsf{V}(\mathsf{PK}, \mathsf{SK}, r', \mathsf{st}[\mathsf{V}]) \wedge \mathsf{m} = \mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK})$ Return $\mathsf{T}$
Return $\mathsf{F}$

**algorithm** $\mathsf{U}'(\mathsf{c}, \mathsf{PK}, \mathsf{R}, \mathsf{m}, r, \mathsf{st}[\mathsf{V}'])$:
$(\mathsf{st}[\mathsf{V}], (\mathsf{SK}^*, \mathsf{PK}^*)) \leftarrow \mathsf{st}[\mathsf{V}']$
$(\mathsf{SK}, r') \leftarrow r$
$\mathsf{R}'(\mathsf{SK}) := \mathsf{R}(\mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK}))$
$\mathsf{st}[\mathsf{V}] \leftarrow \mathsf{U}(\mathsf{PK}, \mathsf{R}', \mathsf{SK}, r', \mathsf{st}[\mathsf{V}])$
Return $(\mathsf{st}[\mathsf{V}], (\mathsf{SK}^*, \mathsf{PK}^*))$

Fig. 5: $\mathsf{U}'$ and $\mathsf{V}'$ for the strong decryption oracle corresponding to $\mathbf{Invert}_{\mathsf{U},\mathsf{V}}$ with $\mathsf{st}_0 = (\mathsf{SK}^\star, \mathsf{PK}^\star)$.

standard decryption oracle is used (and a public key is fixed). We therefore propose strong plaintext awareness as a natural extension for strong security models. This notion, however, is not the only way to render strong decryption oracles ineffective. An alternative approach is to require that any adversary which outputs a new valid public key must know a valid secret key for it. We refer to this property as secret key awareness. In the next two subsections we formalise these extractor-based notions precisely and demonstrate their adequacy for the security analysis of completely non-malleable schemes.

### 3.1 Strong plaintext awareness

We follow the approach adapted in [BP04] to define strong plaintext awareness in the standard model. We run an adversary in two possible environments and require that its behaviour does not change in any significant way. In the first world, the adversary has access to a real strong decryption oracle while in the second the oracle executes a polynomial-time extractor. Furthermore, in these environments, the adversary may obtain ciphertexts on "unknown-but-controlled" plaintexts though an encryption oracle, fed with messages produced by a plaintext creator. More formally, the SPAx advantage of an adversary, for $\mathsf{x} = 1, 2$, against encryption scheme $\Pi$ with respect to plaintext creator $\mathcal{P}$ (mapping bit strings to messages), strong plaintext extractor $\mathcal{K}$, and distinguisher $\mathcal{D}$, is defined by[11]

$$\mathbf{Adv}^{\mathsf{spax}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Dec\text{-}SPAx}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right] - \Pr\left[\mathsf{Ext\text{-}SPAx}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right]$$

where games Dec-SPAx and Ext-SPAx are shown in Figure 6. We say a scheme is SPAx if, for every PPT adversary $\mathcal{A}$, there exists an efficient strong plaintext extractor $\mathcal{K}$ such that, for all distinguishers[12] and plaintext creators, advantage is negligible.

The next theorem, which is proved in Appendix B, shows that the above formulation of strong plaintext-awareness, together with semantic security is enough to achieve strong chosen-ciphertext security.

**Theorem 2** (SPAx $\wedge$ IND-CPA $\Rightarrow$ IND-SCCAx)**.** *Fix a definition of* $\mathbf{SDecrypt}_{\mathsf{U},\mathsf{V}}$ *and let* $\mathcal{A}$ *be an* IND-SCCAx *adversary against* $\Pi$ *in the resulting model. Then there exist an* SPAx *ciphertext creator* $\mathcal{A}_1$*, an* IND-CPA *adversary* $\mathcal{A}_2$*, plaintext creators* $\mathcal{P}_0$*,* $\mathcal{P}_1$*, and distinguishers* $\mathcal{D}_0$*,* $\mathcal{D}_1$ *such that*

$$\mathbf{Adv}^{\mathsf{ind\text{-}sccax}}_{\Pi,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\mathsf{spax}}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{spax}}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{A}_2}(\lambda),$$

*where* $\mathcal{K}$ *is the plaintext extractor for* $\mathcal{A}_1$ *implied by the* SPAx *property of* $\Pi$*.*

As we mentioned in the introduction, the equivalence between indistinguishability under strong chosen-ciphertext security and simulation-based complete non-malleability is established for *assisted* simulators [BF10]. The next theorem shows that using strong plaintext awareness one can strengthen this result to non-assisted simulators[13].

[11] When $\mathsf{x} = 1$ we omit the plaintext creator $\mathcal{P}$ from the subscripts as it is never called.
[12] If unbounded distinguishers are allowed, we get statistical strong plaintext awareness.
[13] Due to space constraints, the SNM definitions are included in Appendix C.

Game $\mathsf{Dec\text{-}SPAx}_{\Pi,\mathcal{P},\mathcal{D}}(\lambda)$

**procedure** $\mathbf{Initialize}(\lambda)$:
$\mathsf{I} \leftarrow_\$ \mathsf{Setup}(1^\lambda)$
$(\mathsf{SK}^\star, \mathsf{PK}^\star) \leftarrow_\$ \mathsf{Gen}(\mathsf{I})$
Choose coins $\mathsf{Rnd}[\mathcal{A}]$ for $\mathcal{A}$
$\mathsf{st}[\mathcal{P}] \leftarrow \epsilon$; $\mathsf{List} \leftarrow []$; $\mathsf{st}[\mathsf{V}] \leftarrow \mathsf{st}_0$
Return $(\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$

**procedure** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$:
Return $\mathbf{SDecrypt}_{\mathsf{U},\mathsf{V}}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$

**procedure** $\mathbf{Encrypt}(Q)$:
$(\mathsf{m}, \mathsf{st}[\mathcal{P}]) \leftarrow_\$ \mathcal{P}(Q, \mathsf{st}[\mathcal{P}])$
$\mathsf{c} \leftarrow_\$ \mathsf{Enc}(\mathsf{m}, \mathsf{PK}^\star)$
$\mathsf{List} \leftarrow (\mathsf{c}, \mathsf{PK}^\star) : \mathsf{List}$
Return $\mathsf{c}$

**procedure** $\mathbf{Finalize}(x)$:
Return $\mathcal{D}(x)$

Game $\mathsf{Ext\text{-}SPAx}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}}(\lambda)$

**procedure** $\mathbf{Initialize}(\lambda)$:
$\mathsf{I} \leftarrow_\$ \mathsf{Setup}(1^\lambda)$
$(\mathsf{SK}^\star, \mathsf{PK}^\star) \leftarrow_\$ \mathsf{Gen}(\mathsf{I})$
Choose coins $\mathsf{Rnd}[\mathcal{A}]$ for $\mathcal{A}$; $\mathsf{List} \leftarrow []$
$\mathsf{st}[\mathcal{P}] \leftarrow \epsilon$; $\mathsf{st}[\mathcal{K}] \leftarrow (\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$
Return $(\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$

**procedure** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$:
$(\mathsf{m}, \mathsf{st}[\mathcal{K}]) \leftarrow_\$ \mathcal{K}(\mathsf{c}, \mathsf{PK}, \mathsf{R}, \mathsf{List}, \mathsf{st}[\mathcal{K}])$
Return $\mathsf{m}$

**procedure** $\mathbf{Encrypt}(Q)$:
$(\mathsf{m}, \mathsf{st}[\mathcal{P}]) \leftarrow_\$ \mathcal{P}(Q, \mathsf{st}[\mathcal{P}])$
$\mathsf{c} \leftarrow_\$ \mathsf{Enc}(\mathsf{m}, \mathsf{PK}^\star)$
$\mathsf{List} \leftarrow (\mathsf{c}, \mathsf{PK}^\star) : \mathsf{List}$
Return $\mathsf{c}$

**procedure** $\mathbf{Finalize}(x)$:
Return $\mathcal{D}(x)$

Fig. 6: The Dec-SPAx and Ext-SPAx games for defining the strong plaintext-awareness of encryption scheme $\Pi$. An adversary $\mathcal{A}$ is legitimate if: 1) R is polynomial-time and if $\mathsf{x} = 1$ it never calls $\mathbf{Encrypt}$; and 2) It never calls $\mathbf{SDecrypt}$ with a tuple $(\mathsf{c}, \mathsf{PK})$ in List.

**Theorem 3** (SPAx $\wedge$ SNM-CPA $\Rightarrow$ **Non-Assisted** SNM-SCCAx)**.** *Fix a definition of* $\mathbf{SDecrypt}_{\mathsf{U},\mathsf{V}}$ *and let* $\mathcal{A}$ *be a* Real-SNM-SCCAx *adversary against* $\Pi$*. Then there exist an* SPAx *ciphertext creator* $\mathcal{A}_1$*, a* Real-SNM-CPA *adversary* $\mathcal{A}_2$*, a plaintext creator* $\mathcal{P}$*, a distinguisher* $\mathcal{D}$*, and a (non-assisted) simulator* $\mathcal{S}$ *such that for all* R

$$\mathbf{Adv}^{\mathsf{snm\text{-}sccax}}_{\Pi,\mathsf{R},\mathcal{S},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\mathsf{spax}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{snm\text{-}cpa}}_{\Pi,\mathsf{R},\mathcal{S}_2,\mathcal{A}_2}(\lambda),$$

*where* $\mathcal{K}$ *is the strong plaintext extractor for* $\mathcal{A}_1$ *implied by the* SPAx *property of* $\Pi$ *and* $\mathcal{S}_2 = \mathcal{S}$ *is the simulator for* $\mathcal{A}_2$ *implied by the* SNM-CPA *security of* $\Pi$*.*

The proof of this theorem, included in Appendix D, proceeds in a different way than that in [BS06] for standard security models. There, a new key-pair is generated to enable the simulator to answer decryption queries, whereas in our proof this is not necessary. As pointed out by Pass et al. [PSV07], the proof in [BS06] relies on the existence of an algorithm for efficiently encrypting all possible outputs of decryption, including special symbol $\perp$. Plaintext awareness in general does not imply that this must be the case, and so our results extend the results in [PSV07]: schemes that do not have the property identified in [PSV07] may still be plaintext aware, and therefore achieve simulation-based non-malleability for non-assisted simulators. However, as shown in [BD08, Theorem 2], if an encryption scheme is PA2 and has an infinite message space, then it is not OW-CPA. This also applies to strong plaintext awareness, and hence no scheme with an infinite message space will be captured by the above theorem.

REMARK. In the above theorem we do not need to restrict the class of relations, in particular to those which are independent of the challenge public key (called lacking relations in [VV08]). This means that through strong plaintext awareness one can improve on the results in [VV08], where this security level can only be achieved at the cost of relation being independent of the common parameters.

REMARK. Using the techniques introduced by Dent [Den06], the scheme in [BF10] might satisfy strong plaintext awareness under an appropriate (bilinear) Diffie-Hellman knowledge assumptions. We leave this and constructing a strongly plaintext-aware scheme in the standard model as an open problem.

### 3.2 Secret key awareness

We now formalise secret key awareness as an alternative route to achieve strong security guarantees. We take a similar approach to plaintext awareness and give an adversary access to an oracle which is either a real *inversion* oracle (as defined in Section 2) or one which uses a polynomial-time secret key extractor. Once again, our requirement is that the behaviour of the adversary is computationally indistinguishable in the two environments. We also provide the adversary with a decryption and a controlled encryption oracle which model the extra auxiliary information that might be useful in producing a new public key. Formally, the SKAx advantage of an adversary $\mathcal{A}$ against encryption scheme $\Pi$ with respect to secret key extractor $\mathcal{K}$, plaintext creator $\mathcal{P}$, and distinguisher $\mathcal{D}$ is defined by[14]

$$\mathbf{Adv}^{\mathsf{skax}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Inv\text{-}SKAx}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right] - \Pr\left[\mathsf{Ext\text{-}SKAx}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right]$$

where games Inv-SKA and Ext-SKA are shown in Figure 7. We say a scheme is SKAx secure if, for every PPT adversary $\mathcal{A}$, there exists an efficient secret key extractor $\mathcal{K}$ such that, for all distinguishers[15] and plaintext creators, advantage is negligible.

Game $\mathsf{Inv\text{-}SKAx}_{\Pi,\mathcal{P},\mathcal{D}}(\lambda)$

**procedure** $\mathbf{Initialize}(\lambda)$:
$\mathsf{I} \leftarrow_\$ \mathsf{Setup}(1^\lambda)$
$(\mathsf{SK}^\star, \mathsf{PK}^\star) \leftarrow_\$ \mathsf{Gen}(\mathsf{I})$
Choose coins $\mathsf{Rnd}[\mathcal{A}]$ for $\mathcal{A}$
$\mathsf{st}[\mathcal{P}] \leftarrow \epsilon$; $\mathsf{List} \leftarrow []$; $\mathsf{st}[\mathsf{V}] \leftarrow \mathsf{st}_0$
Return $(\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$

**procedure** $\mathbf{Invert}(\mathsf{PK}, \mathsf{R})$:
Return $\mathbf{Invert}_{\mathsf{U},\mathsf{V}}(\mathsf{PK}, \mathsf{R})$

**procedure** $\mathbf{Encrypt}(Q)$:
$(\mathsf{m}, \mathsf{st}[\mathcal{P}]) \leftarrow_\$ \mathcal{P}(Q, \mathsf{st}[\mathcal{P}])$
$\mathsf{c} \leftarrow_\$ \mathsf{Enc}(\mathsf{m}, \mathsf{PK}^\star)$
$\mathsf{List} \leftarrow (\mathsf{c}, \mathsf{PK}^\star) : \mathsf{List}$
Return c

**procedure** $\mathbf{Decrypt}(\mathsf{c})$:
$\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{c}, \mathsf{SK}^\star, \mathsf{PK}^\star)$
Return m

**procedure** $\mathbf{Finalize}(x)$:
Return $\mathcal{D}(x)$

Game $\mathsf{Ext\text{-}SKAx}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}}(\lambda)$

**procedure** $\mathbf{Initialize}(\lambda)$:
$\mathsf{I} \leftarrow_\$ \mathsf{Setup}(1^\lambda)$
$(\mathsf{SK}^\star, \mathsf{PK}^\star) \leftarrow_\$ \mathsf{Gen}(\mathsf{I})$
Choose coins $\mathsf{Rnd}[\mathcal{A}]$ for $\mathcal{A}$
$\mathsf{st}[\mathcal{P}] \leftarrow \epsilon$; $\mathsf{st}[\mathcal{K}] \leftarrow (\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$
$\mathsf{List} \leftarrow []$; $\mathsf{List}' \leftarrow []$
Return $(\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$

**procedure** $\mathbf{Invert}(\mathsf{PK}, \mathsf{R})$:
$(\mathsf{SK}, \mathsf{st}[\mathcal{K}]) \leftarrow_\$ \mathcal{K}(\mathsf{PK}, \mathsf{R}, \mathsf{List}, \mathsf{List}', \mathsf{st}[\mathcal{K}])$
Return SK

**procedure** $\mathbf{Encrypt}(Q)$:
$(\mathsf{m}, \mathsf{st}[\mathcal{P}]) \leftarrow_\$ \mathcal{P}(Q, \mathsf{st}[\mathcal{P}])$
$\mathsf{c} \leftarrow_\$ \mathsf{Enc}(\mathsf{m}, \mathsf{PK}^\star)$
$\mathsf{List} \leftarrow (\mathsf{c}, \mathsf{PK}^\star) : \mathsf{List}$
Return c

**procedure** $\mathbf{Decrypt}(\mathsf{c})$:
$\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{c}, \mathsf{SK}^\star, \mathsf{PK}^\star)$
$\mathsf{List}' \leftarrow \mathsf{m} : \mathsf{List}'$
Return m

**procedure** $\mathbf{Finalize}(x)$:
Return $\mathcal{D}(x)$

Fig. 7: The Inv-SKAx and Ext-SKAx games for defining secret key awareness. An adversary $\mathcal{A}$ is legitimate if: 1) R is polynomial-time and, if $\mathsf{x} = 0$ it never calls $\mathbf{Decrypt}$ or $\mathbf{Encrypt}$ and if $\mathsf{x} = 1$ it never calls $\mathbf{Encrypt}$; 2) It never queries $\mathsf{PK}^\star$ to $\mathbf{Invert}$; and 3) It never calls $\mathbf{Decrypt}$ with a ciphertext c such that $(\mathsf{c}, \mathsf{PK}^\star) \in \mathsf{List}$.

We are now ready to state the main theorem of this section, which permits concluding that secret key awareness combined with IND-CCA2 is strong enough to guarantee IND-SCCA2 security. The proof is analogous to that of Theorem 2 and is included in Appendix E.

**Theorem 4** (SKAx$\wedge$IND-CCAx $\Rightarrow$ IND-ICAx)**.** *Fix a definition of* $\mathbf{Invert}_{\mathsf{U},\mathsf{V}}$ *and let* $\mathcal{A}$ *be an* IND-ICAx *adversary against* $\Pi$*. Then, there exist an* SKAx *public key creator* $\mathcal{A}_1$*, an* IND-CCAx *adversary* $\mathcal{A}_2$*, plaintext creators* $\mathcal{P}_0$*,* $\mathcal{P}_1$*, and distinguishers* $\mathcal{D}_0$*,* $\mathcal{D}_1$ *such that*

$$\mathbf{Adv}^{\mathsf{ind\text{-}icax}}_{\Pi,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\mathsf{skax}}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{skax}}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ind\text{-}ccax}}_{\Pi,\mathcal{A}_2}(\lambda),$$

[14] Once again, when $\mathsf{x} = 0, 1$ we omit the plaintext creator $\mathcal{P}$ from the subscripts as it is never called.

[15] If unbounded distinguishers are allowed, we get statistical secret key awareness.

*where $\mathcal{K}$ is the secret key extractor for $\mathcal{A}_1$ implied by the* SKAx *property of* $\Pi$.

To further justify the definition of secret key awareness, we show that it can be used to achieve strong plaintext awareness. The next theorem, proved in in Appendix F, states that secret key awareness combined with standard plaintext awareness gives rise to strong plaintext awareness.

**Theorem 5** **(SKAx$\wedge$PAx $\Rightarrow$ SPAx).** *Fix a definition of* $\mathbf{Invert}_{\mathsf{U},\mathsf{V}}$ *and let $\mathcal{A}$ be an* SPAx *ciphertext creator against* $\Pi$*, with respect to the* $\mathbf{SDecrypt}_{\mathsf{U}',\mathsf{V}'}$ *procedure associated to* $\mathbf{Invert}_{\mathsf{U},\mathsf{V}}$ *as defined in Figure 5. Then there exists an* SKAx *public key creator $\mathcal{A}_1$, a* PAx *ciphertext creator $\mathcal{A}_2$, and an* SPAx *plaintext extractor $\mathcal{K}$ such that for any plaintext creator $\mathcal{P}$, and any distinguisher $\mathcal{D}$ we have*

$$\mathbf{Adv}^{\mathsf{spax}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\mathsf{skax}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}_1\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{pax}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}_2,\mathcal{A}_2}(\lambda),$$

*where $\mathcal{K}_1$ is the a secret key extractor for $\mathcal{A}_1$ implied by the* SKAx *property of* $\Pi$*, and $\mathcal{K}_2$ is the plaintext extractor for $\mathcal{A}_2$ implied by the* PAx *property of* $\Pi$.

The intuition behind this theorem is the following. Secret key awareness ensures that a strong plaintext awareness adversary cannot come up with a ciphertext under a *new* public key, for which it does not know the underlying message (as it must know the decryption key). However, no such guarantee is provided for the *challenge* public key, and this justifies the plaintext awareness requirement.

REMARK. An extra feature that comes with secret key awareness is the ability to open ciphertexts via the secret key[16]. In other words, one can convert a non-malleability simulator that only returns $(\mathsf{PK}^\star, \mathsf{c}^\star)$ to another one[17] which also outputs the corresponding opening $(\mathsf{SK}^\star, \mathsf{m}^\star)$. This means that the output of the simulator can indeed be seen as a de-commitment. The same observation does not apply to strong plaintext awareness, as this notion does not guarantee the knowledge of the *randomness* used in encryption.

## 4 Secret key aware schemes

### 4.1 Generic construction with a random oracle

We have defined strong plaintext and secret key awareness in the standard model, but the definitions can be adapted to the random oracle model [BR93] in the natural way[18]. Interestingly, in the random oracle model, there is a simple transformation that turns any encryption scheme into one which is secret key aware without any loss in security: one just changes the key-generation algorithm by attaching the hash of the key-pair to the public key. More formally, the transformed set-up algorithm $\mathsf{Setup}'$ is identical to $\mathsf{Setup}$ except that it also specifies a new independent hash function $\mathsf{H}$ (i.e. one which is not used by the scheme), which will be modelled as a random oracle in the security analysis. The remaining algorithms are shown in Figure 8.

| **procedure** $\mathsf{Gen}'(\mathsf{I})$: | **procedure** $\mathsf{Enc}'(\mathsf{m}, \mathsf{PK}')$: | **procedure** $\mathsf{Dec}'(\mathsf{c}, \mathsf{SK}', \mathsf{PK}')$: |
|---|---|---|
| $(\mathsf{SK}, \mathsf{PK}) \leftarrow_{\$} \mathsf{Gen}(\mathsf{I})$ | $(\mathsf{PK}, h) \leftarrow \mathsf{PK}'$ | $(\mathsf{PK}, h) \leftarrow \mathsf{PK}'$; $\mathsf{SK} \leftarrow \mathsf{SK}'$ |
| $\mathsf{PK}' \leftarrow (\mathsf{PK}, \mathsf{H}(\mathsf{SK}, \mathsf{PK}))$ | $\mathsf{c} \leftarrow_{\$} \mathsf{Enc}(\mathsf{m}, \mathsf{PK})$ | If $h \neq \mathsf{H}(\mathsf{SK}, \mathsf{PK})$ Return $\perp$ |
| Return $(\mathsf{SK}, \mathsf{PK}')$ | Return $\mathsf{c}$ | Return $\mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK})$ |

Fig. 8: Generic transformation to a secret-key-aware scheme $\Pi'$ in the random oracle model.

[16] Although not considered in this paper, the secret key awareness property can also be used in composition theorems where direct access to secret keys is required. This could be useful, for example, in signcryption.

[17] This new simulator must be given the secret key for the challenge public key, which is consistent with the notion of a non-malleable commitment.

[18] Furthermore, the standard stronger definition requiring the existence of a universal extractor may also be easily formulated [BDPR98].

It can be easily shown that the scheme obtained through the above transformation is secret key aware, as long as the range of $\mathsf{H}$ is large enough to ensure that the transformed scheme has only one valid secret key for each valid public key with overwhelming probability. The idea behind the proof is that adversarially created public keys will be invalid with high probability, unless the random oracle has already been queried on the corresponding secret key. In this case the (unique) secret key can be recovered using the extractability property of the random oracle. Note that simply attaching $\mathsf{H}(\mathsf{SK})$ is not enough, as extraction will fail in case the challenge public key is malleable. See the details in Appendix G. From this result, together with Theorems 1 and 4, we can deduce that if $\Pi$ is IND-CCAx-secure then $\Pi'$ is IND-SCCAx-secure.

REMARK. The previous generic construction can be applied to RSA-OAEP. In the random oracle model, this new version of RSA-OAEP is SKA2 because of the modified keys, and it preserves the original PA2 and IND-CPA security of RSA-OAEP. It follows that this scheme is completely non-malleable with respect to non-assisted simulators. Note that we do not need to restrict the adversary to querying only valid public keys, as is the case in [Fis05], since the secret key extractor implied by the SKA2 property will permit detecting such invalid queries.

### 4.2 Towards secret-key-aware schemes without random oracles

We present two approaches to constructing secret key aware schemes without random oracles. Both are intended as stepping stones towards achieving the strongest forms of secret key awareness. We first introduce a new knowledge-based assumption and use it to construct a concrete scheme, which is "weakly" secret key aware. Then we propose a generic construction inspired by techniques used in encryption schemes with key escrow [BNB07]. We leave it as an interesting open problem to instantiate this generic construction or show its unrealisability.

THE KNOWLEDGE OF FACTOR ASSUMPTION. We take advantage of the fact that $k$-bit integers of the form $N = P^2Q$ have a negligible density in the set of all $k$-bit integers (note that this is *not* the case for the integers of the form $PQ$), and we postulate that the only way to generate such integers is to start with the two prime factors and calculate $N$. This assumption is similar to Diffie-Hellman knowledge type assumptions [BP04] where one exploits the sparse image of the $r \mapsto (g^r, (g^a)^r)$ map. Our assumption, however, has the extra property of being "non-malleable" in the sense that there does not seem to be any way to use the knowledge of one (or in fact many) integers of this form to find an alternative way to construct new ones. Diffie-Hellman tuples on the other hand are malleable. For concreteness, we now present a formal definition of our *knowledge of factorisation assumptions*.

Take $\mathcal{G}_e$ to be the algorithm that, for a given value of the security parameter, generates numbers of the form $P^2Q$, with $P$ and $Q$ random primes of the appropriate size such that[19] $\gcd(e,\varphi(P^{\star 2}Q^{\star})) = 1$. Figure 9 depicts the KFAx game for $\mathsf{x} = 0, 1, 2$, where an adversary is required to construct a new integer of the same form *without knowing* the factorization. We define the KFAx advantage of an adversary against $\mathcal{G}_e$, with respect to knowledge extractor $\mathcal{K}$ as:

$$\mathbf{Adv}^{\mathsf{kfax}}_{\mathcal{G}_e,\mathcal{K},\ell,\mathcal{A}}(\lambda) := \Pr[\mathsf{KFAx}^{\mathcal{A}}_{\mathcal{G}_e,\mathcal{K},\ell}(\lambda) \Rightarrow \mathsf{T}].$$

The KFAx assumption states that, for every PPT adversary, there exists an efficient knowledge extractor such that advantage is negligible.

RSA-BASED SECRET-KEY-AWARE SCHEMES. The KFA1 assumption immediately implies that an RSA-based scheme with $P^2Q$ modulus [Tak98] is SKA1. Random padding before encryption allows one to construct an IND-CPA secure scheme. In order to extend this to IND-CCA1 security, a non-adaptive $\mathbf{Root}$ oracle is added to the RSA problem. As a result, we arrive at an IND-SCCA1 secure encryption scheme without random oracles and with no setup assumptions. We refer the reader to Appendix H for the details on this concrete scheme and a proof of the secret key awareness property.

The only factorisation/RSA-based IND-CCA2 secure encryption scheme in the standard model is a recent scheme of Hofheinz and Kiltz [HK09]. This scheme, with appropriately modified public keys is a candidate for

[19] We use $\varphi$ to denote Euler's totient function.

Game $\mathsf{KFAx}_{\mathcal{G}_e,\mathcal{K},\ell}(\lambda)$

**procedure** $\mathbf{Initialize}(\lambda)$:
$(P^\star, Q^\star) \leftarrow_\$ \mathcal{G}_e(1^\lambda)$; $N^\star \leftarrow {P^\star}^2 Q^\star$
$d \leftarrow 1/e \pmod{\varphi(N^\star)}$; $\mathsf{List} \leftarrow []$
Choose $\mathsf{Rnd}[\mathcal{A}]$ for $\mathcal{A}$; $\mathsf{Flag} \leftarrow \mathsf{F}$
$\mathsf{st}[\mathcal{K}] \leftarrow (N^\star, \mathsf{Rnd}[\mathcal{A}])$
Return $(N^\star, \mathsf{Rnd}[\mathcal{A}])$

**procedure** $\mathbf{Root}(y)$:
$t \leftarrow y^d \pmod{N^*}$; $(x, x') \leftarrow t$
$\mathsf{List} \leftarrow (x, y) : \mathsf{List}$
Return $x$

**procedure** $\mathbf{Factor}(N)$:
$((P, Q), \mathsf{st}[\mathcal{K}]) \leftarrow_\$ \mathcal{K}(N, \mathsf{List}, \mathsf{st}[\mathcal{K}])$
If $P^2 Q = N \wedge P \neq 1$ Return $(P, Q)$
If $\exists P', Q'$ s.t. ${P'}^2 Q' = N$
  Set $\mathsf{Flag} \leftarrow \mathsf{T}$
Return $(\perp, \perp)$

**procedure** $\mathbf{Finalize}()$:
Return $\mathsf{Flag}$

Fig. 9: Game defining the knowledge of factor assumption. An adversary $\mathcal{A}$ is legitimate if: 1) If $\mathsf{x} = 0$ it queries $\mathbf{Factor}$ once, and if $\mathsf{x} = 0, 1$ it does not query $\mathbf{Root}$; and 2) It never queries $\mathbf{Factor}$ on $N^\star$. $\mathbf{Root}$ returns the first $\ell$ bits of $t$, i.e. $|x| = \ell$.

achieving IND-SCCA2 security under the KFAx assumptions through secret key awareness. Such a construction would also admit a (non-black-box) non-assisted complete non-malleability simulator *with no set-up assumptions*. This would solve the open problem [Fis05] of constructing an encryption scheme that is suitable for the implementation of non-malleable commitment schemes in the plain model.

REMARK. The KFAx assumptions lead to a construction of extractable one-way functions analogous to that obtained using the knowledge-of-exponent assumptions [CD08,CD09]. However, we can use even the weakest form KFA0 to go beyond. Indeed, this assumption states that one cannot come up with an $N$ of the correct form, even if *given another* integer of this form as auxiliary information. Under this assumption the function $f(P, Q) = P^2 Q$, where $P$ and $Q$ are $k$-bit primes, is an extractable one-way function with *dependent* auxiliary information.

REMARK. We note that knowledge assumptions seem necessary to establish plaintext and secret key awareness. It remains an open problem to construct plaintext-aware schemes without relying on extractor-based assumptions such as Diffie-Hellman Knowledge. NIZK techniques do not provide an answer to this problem, as extractors should work with the *provided* common reference string.

A GENERIC TECHNIQUE BASED ON SCHEMES WITH KEY ESCROW. Consider a public-key encryption scheme $\Pi$ where the key-generation procedure first generates a secret key in the appropriate range, and then encrypts it under an auxiliary encryption scheme[20] $\Pi_{\mathsf{PK}}$. Then, the plaintext awareness property of $\Pi_{\mathsf{PK}}$ naturally maps to (a weak form of) secret key awareness for $\Pi$. The caveat to this design technique is that as shown in [TO08b] plaintext awareness is an all-or-nothing notion, which could render this construction unrealisable. Indeed, full plaintext awareness in key-generation would imply a form of indistinguishability for secret keys that is contradicted by the correctness of the scheme[21]. However, we find that by restricting the plaintext awareness property of $\Pi_{\mathsf{PK}}$ to the class of plaintext creators that return a random message from the message space, we can get around this obstacle. Indeed, in this case it could be possible to show that the resulting scheme achieves secret key awareness.

## Acknowledgments

The authors were funded in part by eCrypt II (EU FP7 - ICT-2007-216646) and FCT project PTDC/EIA/71362/2006. The second author was also funded by FCT grant BPD-47924-2008. Research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Gov-

[20] An auxiliary KEM would be more efficient. Here we only aim to present the intuition.
[21] Given a public key and two secret keys one can check which secret key is valid through the encryption and decryption algorithms.

ernment, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

## References


ARP03. Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless public key cryptography. In Chi-Sung Laih, editor, *ASIACRYPT*, volume 2894 of *LNCS*, pages 452–473. Springer, 2003.

BBM00. Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *LNCS*, pages 259–274. Springer, 2000.

BD08. James Birkett and Alexander W. Dent. Relations among notions of plaintext awareness. In Cramer [Cra08], pages 47–64.

BDPR98. Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Krawczyk [Kra98].

BF10. Manuel Barbosa and Pooya Farshim. Relations among notions of complete non-malleability: Indistinguishability characterisation and efficient construction without random oracles. Pre-print (accepted for ACISP 2010), 2010.

BNB07. Jaimee Brown, Juan M. Gonzalez Nieto, and Colin Boyd. Efficient and secure self-escrowed public-key infrastructures. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 284–294, New York, NY, USA, 2007. ACM.

BP04. Mihir Bellare and Adriana Palacio. Towards plaintext-aware public-key encryption without random oracles. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *LNCS*, pages 48–62. Springer, 2004.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

BR94. Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *EUROCRYPT*, volume 950 of *LNCS*, pages 92–111. Springer, 1994.

BR06. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Vaudenay [Vau06], pages 409–426.

BS99. Mihir Bellare and Amit Sahai. Non-malleable encryption: Equivalence between two notions, and an indistinguishability-based characterization. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *LNCS*, pages 519–536. Springer, 1999.

BS06. Mihir Bellare and Amit Sahai. Non-malleable encryption: Equivalence between two notions, and an indistinguishability-based characterization. Cryptology ePrint Archive, Report 2006/228, 2006. `http://eprint.iacr.org/2006/228`.

CD08. Ran Canetti and Ronny Ramzi Dakdouk. Extractable perfectly one-way functions. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (2)*, volume 5126 of *LNCS*, pages 449–460. Springer, 2008.

CD09. Ran Canetti and Ronny Ramzi Dakdouk. Towards a theory of extractable functions. In Omer Reingold, editor, *TCC*, volume 5444 of *LNCS*. Springer, 2009.

Cra08. Ronald Cramer, editor. *Public Key Cryptography - PKC 2008, 11th International Workshop on Practice and Theory in Public-Key Cryptography, Barcelona, Spain, March 9-12, 2008. Proceedings*, volume 4939 of *LNCS*. Springer, 2008.

DDN91. Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *STOC*, pages 542–552. ACM, 1991.

DDN00. Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000.

Den06. Alexander W. Dent. The cramer-shoup encryption scheme is plaintext aware in the standard model. In Vaudenay [Vau06], pages 289–307.

Fis05. Marc Fischlin. Completely non-malleable schemes. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP*, volume 3580 of *LNCS*, pages 779–790. Springer, 2005.

Fuj06. Eiichiro Fujisaki. Plaintext simulatability. *IEICE Transactions*, 89-A(1):55–65, 2006.

Gol04. Oded Goldreich. *Foundations of Cryptography: Volume II Basic Applications*. Cambridge University Press, 2004.

HK09. Dennis Hofheinz and Eike Kiltz. Practical chosen ciphertext secure encryption from factoring. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *LNCS*, pages 313–332. Springer, 2009.

HLM03. Jonathan Herzog, Moses Liskov, and Silvio Micali. Plaintext awareness via key registration. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *LNCS*, pages 548–564. Springer, 2003.

Kra98. Hugo Krawczyk, editor. *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *LNCS*. Springer, 1998.

PPV08. Omkant Pandey, Rafael Pass, and Vinod Vaikuntanathan. Adaptive one-way functions and applications. In David Wagner, editor, *CRYPTO*, volume 5157 of *LNCS*, pages 57–74. Springer, 2008.

PSV07. Rafael Pass, Abhi Shelat, and Vinod Vaikuntanathan. Relations among notions of non-malleability for encryption. In *ASIACRYPT*, pages 519–535, 2007.

RGK06. Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Deniable authentication and key exchange. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 400–409. ACM, 2006.

Tak98. Tsuyoshi Takagi. Fast RSA-type cryptosystem modulo $p^k q$. In Krawczyk [Kra98].

TO08a. Isamu Teranishi and Wakaha Ogata. Cramer-shoup satisfies a stronger plaintext awareness under a weaker assumption. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN*, volume 5229 of *LNCS*. Springer, 2008.

TO08b. Isamu Teranishi and Wakaha Ogata. Relationship between standard model plaintext awareness and message hiding. *IEICE Transactions*, 91-A(1):244–261, 2008.

TO08c. Isamu Teranishi and Wakaha Ogata. Relationship between two approaches for defining the standard model pa-ness. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *ACISP*, volume 5107 of *LNCS*, pages 113–127. Springer, 2008.

Vau06. Serge Vaudenay, editor. *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *LNCS*. Springer, 2006.

VV08. Carmine Ventre and Ivan Visconti. Completely non-malleable encryption revisited. In Cramer [Cra08], pages 65–84.

## A Proof of Theorem 1: IND-ICAx $\Rightarrow$ IND-SCCAx

*Proof.* Let $\mathcal{A}$ be an IND-SCCAx adversary against $\Pi$ with respect to the decryption oracle defined in Figure 5. We construct an IND-ICAx adversary $\mathcal{A}_1$ against $\Pi$ as shown in Figure 10 and show that this simulates the oracle defined in Figure 5 perfectly. This is straightforward when the queried public key to strong decryption oracle is the challenge one: in both cases a standard decryption oracle is used and the message is returned if $\mathsf{R}$ holds. We analyse the $\mathsf{PK} \neq \mathsf{PK}^*$ case next.

**adversary** $\mathcal{A}_1$:
Run $\mathcal{A}$ in the environment below

$\mathbf{Initialize}(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star)$ from $\mathbf{Initialize}(\lambda)$
Return $(\mathsf{I}, \mathsf{PK}^\star)$

**query** $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$:
Query $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$ to get $\mathsf{c}$
Return $\mathsf{c}$

**query** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$:
If $\mathsf{PK} = \mathsf{PK}^\star$
  Query $\mathbf{Decrypt}(\mathsf{c})$ to get $\mathsf{m}$
  If $\mathsf{R}(\mathsf{m})$ Return $\mathsf{m}$ Else Return $\perp$
$\mathsf{R}'(\mathsf{SK}) := \mathsf{R}(\mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK}))$
Query $\mathbf{Invert}(\mathsf{PK}, \mathsf{R}')$ to get $\mathsf{SK}$
If $\mathsf{SK} = \perp$ Return $\perp$ Else $\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK})$
Return $\mathsf{m}$

**query** $\mathbf{Finalize}(d)$:
Query $\mathbf{Finalize}(d)$

Fig. 10: An IND-ICAx adversary based on a IND-SCCAx adversary.

Let us first look at how the simulated $\mathbf{SDecrypt}$ operates. It first defines a relation $\mathsf{R}'$, which specifies that a secret key is of interest if $\mathsf{c}$ decrypts to $\mathsf{m}$ satisfying $\mathsf{R}$. Then it queries $\mathbf{Invert}$ to obtain a random secret key from the set of all secret keys which satisfy $\mathsf{R}'$ (all of which corresponding to $\mathsf{PK}$, as otherwise decryption would fail), and also satisfy $\mathsf{V}$ inside $\mathbf{Invert}$. The ciphertext is decrypted with respect to this $\mathsf{SK}$ and returned. Note that $\mathsf{R}(\mathsf{m})$ holds because of the way $\mathsf{R}'$ is defined. We now observe that, for a fixed ciphertext and public key, $\mathsf{SK}$ uniquely determines the decrypted message $\mathsf{m}$. This implies that the distribution of the secret key returned by $\mathbf{Invert}$ induces a distribution on the returned message, which is equivalent to sampling uniformly at random from the set of all $(\mathsf{SK}, \mathsf{m}, r)$ tuples that satisfy the relation:

$$\bar{\mathsf{R}}(\mathsf{SK}, \mathsf{m}, r) := \mathsf{V}(\mathsf{PK}, \mathsf{SK}, r, \mathsf{st}[\mathsf{V}]) \wedge \mathsf{m} = \mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK})) \wedge \mathsf{R}(\mathsf{m}).$$

Note that in such tuples it is possible that $\mathsf{m} = \perp$, as long as the relation $\mathsf{R}$ allows this.

Turning to Figure 5, let us look at how $\mathsf{WitSp}$ is constructed using the validity algorithm $\mathsf{V}'$. The validity algorithm first parses the random string passed to it into a secret key and a randomness. These are used as the arguments to the same $\mathsf{V}$ inside $\mathbf{Invert}$. Doing so ensures that the range of $(\mathsf{SK}, r')$ pairs fed to algorithm $\mathsf{V}$ is identical to that described in the previous paragraph. Furthermore, of these pairs, only those which contain an $\mathsf{SK}$ that correctly decrypts $\mathsf{m}$ will be allowed to pass (note that this includes the case where $\mathsf{m} = \perp$). As a result, the witness space will contain a list of $(\mathsf{m}, r)$ pairs which maps, one-to-one, to an equivalent list of triples $(\mathsf{m}, \mathsf{SK}, r')$, all of them satisfying relation

$$\hat{\mathsf{R}}(\mathsf{SK}, \mathsf{m}, r') := \mathsf{V}(\mathsf{PK}, \mathsf{SK}, r', \mathsf{st}[\mathsf{V}]) \wedge \mathsf{m} = \mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK})).$$

The returned message will be sampled uniformly at random from those tuples in this witness set that also satisfy $\mathsf{R}(\mathsf{m})$. This means that $\mathsf{m}$ is sampled from the exact same distribution as that described via $\bar{\mathsf{R}}$ above. □

## B Proof of Theorem 2: SPAx $\wedge$ IND-CPA $\Rightarrow$ IND-SCCAx

*Proof.* The proof of this theorem follows that of [BP04, Theorem 4.1]. We first prove the theorem for $\mathsf{x} = 1$. To this end, let $\mathcal{A}$ be an IND-SCCA1 adversary against $\Pi$. We construct a ciphertext creator $\mathcal{A}_1$ against $\Pi$ based on

$\mathcal{A}$, as shown on the top left in Figure 11, and let $\mathcal{K}$ be a plaintext extractor for $\mathcal{A}_1$. We then construct an IND-CPA adversary $\mathcal{A}_2$ against $\Pi$ based on $\mathcal{A}$ and $\mathcal{K}$ (shown in Figure 11 on the right). We also define two distinguishers $\mathcal{D}_0$ and $\mathcal{D}_1$, on the bottom left of Figure 11, and show that:

$$\mathbf{Adv}^{\mathsf{ind\text{-}scca1}}_{\Pi,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\mathsf{spa1}}_{\Pi,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{spa1}}_{\Pi,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{A}_2}(\lambda).$$

**Claim B1.** *We have:*

$$\begin{aligned}
\Pr\left[\mathsf{IND\text{-}SCCA1}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] &= \Pr\left[\mathsf{Dec\text{-}SPA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_1}(\lambda) \Rightarrow \mathsf{T}\right] \\
\Pr\left[\mathsf{IND\text{-}SCCA1}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] &= 1 - \Pr\left[\mathsf{Dec\text{-}SPA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_0}(\lambda) \Rightarrow \mathsf{T}\right] \\
\Pr\left[\mathsf{IND\text{-}CPA}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] &= \Pr\left[\mathsf{Ext\text{-}SPA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_1,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right] \\
\Pr\left[\mathsf{IND\text{-}CPA}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] &= 1 - \Pr\left[\mathsf{Ext\text{-}SPA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_0,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right]
\end{aligned}$$

We prove claims in the end of this section. The result now follows once we use this claim to observe that:

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{ind\text{-}scca1}}_{\Pi,\mathcal{A}}(\lambda) &= \Pr\left[\mathsf{IND\text{-}SCCA1}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] - \Pr\left[\mathsf{IND\text{-}SCCA1}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] \\
&= \Pr\left[\mathsf{Dec\text{-}SPA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_1}(\lambda) \Rightarrow \mathsf{T}\right] - \left(1 - \Pr\left[\mathsf{Dec\text{-}SPA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_0}(\lambda) \Rightarrow \mathsf{T}\right]\right) \\
&= \Pr\left[\mathsf{Ext\text{-}SPA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_1,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right] - \left(1 - \Pr\left[\mathsf{Ext\text{-}SPA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_0,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right]\right) \\
&\quad + \mathbf{Adv}^{\mathsf{spa1}}_{\Pi,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{spa1}}_{\Pi,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) \\
&= \Pr\left[\mathsf{IND\text{-}CPA}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] - \Pr\left[\mathsf{IND\text{-}CPA}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] \\
&\quad + \mathbf{Adv}^{\mathsf{spa1}}_{\Pi,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{spa1}}_{\Pi,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) \\
&= \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{A}_2}(\lambda) + \mathbf{Adv}^{\mathsf{spa1}}_{\Pi,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{spa1}}_{\Pi,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda).
\end{aligned}$$

We now prove the theorem for $\mathsf{x} = 2$. Let $\mathcal{A}$ be an IND-SCCA2 against $\Pi$. We construct a ciphertext creator $\mathcal{A}_1$ against $\Pi$ based on $\mathcal{A}$, as shown on the top left of Figure 12, and let $\mathcal{K}$ be a plaintext extractor for $\mathcal{A}_1$. We then construct an IND-CPA adversary $\mathcal{A}_2$ against $\Pi$ based on $\mathcal{A}$ and $\mathcal{K}$ (shown in Figure 12 on the right). Finally we give two plaintext creators $\mathcal{P}_0$ and $\mathcal{P}_1$ and two distinguishers $\mathcal{D}_0$ and $\mathcal{D}_1$, both shown on the bottom left of Figure 12, such that:

$$\mathbf{Adv}^{\mathsf{ind\text{-}scca2}}_{\Pi,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\mathsf{spa2}}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{spa2}}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{A}_2}(\lambda).$$

**Claim B2.** *We have:*

$$\begin{aligned}
\Pr\left[\mathsf{IND\text{-}SCCA2}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] &= \Pr\left[\mathsf{Dec\text{-}SPA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_1,\mathcal{D}_1}(\lambda) \Rightarrow \mathsf{T}\right] \\
\Pr\left[\mathsf{IND\text{-}SCCA2}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] &= 1 - \Pr\left[\mathsf{Dec\text{-}SPA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_0,\mathcal{D}_0}(\lambda) \Rightarrow \mathsf{T}\right] \\
\Pr\left[\mathsf{IND\text{-}CPA}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] &= \Pr\left[\mathsf{Ext\text{-}SPA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right] \\
\Pr\left[\mathsf{IND\text{-}CPA}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] &= 1 - \Pr\left[\mathsf{Ext\text{-}SPA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right]
\end{aligned}$$

The result follows from the above claim once we observe:

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{ind\text{-}scca2}}_{\Pi,\mathcal{A}}(\lambda) &= \Pr\left[\mathsf{IND\text{-}SCCA2}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] - \Pr\left[\mathsf{IND\text{-}SCCA2}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] \\
&= \Pr\left[\mathsf{Dec\text{-}SPAs}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_1,\mathcal{D}_1}(\lambda) \Rightarrow \mathsf{T}\right] - \left(1 - \Pr\left[\mathsf{Dec\text{-}SPA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_0,\mathcal{D}_0}(\lambda) \Rightarrow \mathsf{T}\right]\right) \\
&= \Pr\left[\mathsf{Ext\text{-}SPA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right] - \left(1 - \Pr\left[\mathsf{Ext\text{-}SPA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right]\right) \\
&\quad + \mathbf{Adv}^{\mathsf{spa2}}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{spa2}}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) \\
&= \Pr\left[\mathsf{IND\text{-}CPA}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] - \Pr\left[\mathsf{IND\text{-}CPA}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] \\
&\quad + \mathbf{Adv}^{\mathsf{spa2}}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{spa2}}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) \\
&= \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{A}_2}(\lambda) + \mathbf{Adv}^{\mathsf{spa2}}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{spa2}}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda).
\end{aligned}$$

□

**adversary** $\mathcal{A}_1$:
Run $\mathcal{A}$ in the environment below

$\mathbf{Initialize}(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_1])$ from $\mathbf{Initialize}(\lambda)$
Return $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_1])$

**query** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$:
Query $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$ to get $\mathsf{m}$
Return $\mathsf{m}$

**query** $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$:
$x \leftarrow (\mathsf{m}_0, \mathsf{m}_1, \mathsf{PK}^\star)$
Query $\mathbf{Finalize}(x)$

**distinguisher** $\mathcal{D}_i(x)$:
$(\mathsf{m}_0, \mathsf{m}_1, \mathsf{PK}^\star) \leftarrow x$
$\mathsf{c} \leftarrow_\$ \mathsf{Enc}(\mathsf{m}_i, \mathsf{PK}^\star)$
Resume $\mathcal{A}(\mathsf{c})$ to get $d$
Return $d \oplus \neg i$

**adversary** $\mathcal{A}_2$:
Run $\mathcal{A}$ in the environment below

$\mathbf{Initialize}(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_2])$ from $\mathbf{Initialize}(\lambda)$
$(\mathsf{Rnd}[\mathcal{A}], \mathsf{Rnd}[\mathcal{K}]) \leftarrow \mathsf{Rnd}[\mathcal{A}_2]$
$\mathsf{st}[\mathcal{K}] \leftarrow (\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$
$\mathsf{List} \leftarrow []$
Return $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}])$

**query** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$:
$(\mathsf{m}, \mathsf{st}[\mathcal{K}]) \leftarrow_\$ \mathcal{K}(\mathsf{c}, \mathsf{PK}, \mathsf{R}, \mathsf{List}, \mathsf{st}[\mathcal{K}]; \mathsf{Rnd}[\mathcal{K}])$
Return $\mathsf{m}$

**query** $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$:
Query $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$ to get $\mathsf{c}$
$\mathsf{List} \leftarrow (\mathsf{c}, \mathsf{PK}^\star) : \mathsf{List}$ Return $\mathsf{c}$

**query** $\mathbf{Finalize}(d)$:
Query $\mathbf{Finalize}(d)$

Fig. 11: SPA1 ciphertext creator $\mathcal{A}_1$, distinguishers $\mathcal{D}_0$ and $\mathcal{D}_1$, and IND-CPA adversary $\mathcal{A}_2$.

*Proof (Claims B1 and B2).* The proofs of these statements are identical to the analogous claims in [BP04]. We give the details only for the first equality in Claim B1. Let $S_{\mathsf{cca1\text{-}1}}$ denote the sample space underlying $\mathsf{IND\text{-}SCCA1}^{\mathcal{A}}_{\Pi}$ when $b = 1$. A member of this space is a string specifying coin tosses for all algorithms involved, which in this case means the coins of the key-generation algorithm, the random tape of $\mathcal{A}$ itself, and the coins used by the encryption algorithm. A member of the sample space $S_{\mathsf{dec\text{-}spa1}}$ underlying $\mathsf{Dec\text{-}SPA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_1}$ is a string specifying the coins of the key-generation algorithm, the random tape of $\mathcal{A}_1$, and the random tape of $\mathcal{D}_1$. The first part of the claim follows once we observe that by the definitions of $\mathcal{A}_1$ and $\mathcal{D}_1$, $S_{\mathsf{dec\text{-}spa1}}$ is equal to $S_{\mathsf{cca1\text{-}1}}$ (the random tape of $\mathcal{A}_1$ consists of coins for $\mathcal{A}$, and the random tape of $\mathcal{D}_1$ consists of coins for the encryption algorithm), and that when $b = 1$, game $\mathsf{IND\text{-}SCCA1}^{\mathcal{A}}_{\Pi}$ returns $\mathsf{T}$ if and only if $\mathsf{Dec\text{-}SPA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_1}$ returns $\mathsf{T}$. □

## C Simulation-based complete non-malleability

The SNM-SCCAx advantage of an adversary $\mathcal{A}$ with respect to a polynomial-time relation $\mathsf{R}$ and a polynomial-time simulator $\mathcal{S}$ against a public-key encryption scheme $\Pi$ is defined by

$$\mathbf{Adv}^{\mathsf{snm\text{-}sccax}}_{\Pi,\mathsf{R},\mathcal{S},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Real\text{-}SNM\text{-}SCCAx}^{\mathcal{A}}_{\Pi,\mathsf{R}}(\lambda) \Rightarrow \mathsf{T}\right] - \Pr\left[\mathsf{Ideal\text{-}SNM\text{-}SCCAx}^{\mathcal{S}}_{\Pi,\mathsf{R}}(\lambda) \Rightarrow \mathsf{T}\right]$$

**adversary** $\mathcal{A}_1$:
Run $\mathcal{A}$ in the environment below

$\mathbf{Initialize}(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_1])$ from $\mathbf{Initialize}(\lambda)$
Return $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_1])$

**query** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$:
Query $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$ to get $\mathsf{m}$
Return $\mathsf{m}$

**query** $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$:
Query $\mathbf{Encrypt}(\mathsf{m}_0, \mathsf{m}_1)$ to get $\mathsf{c}$
Return $\mathsf{c}$

**query** $\mathbf{Finalize}(d)$:
Query $\mathbf{Finalize}(d)$

**plaintext creator** $\mathcal{P}_i(Q, \mathsf{st}[\mathcal{P}_i])$:
$(\mathsf{m}_0, \mathsf{m}_1) \leftarrow Q$
Return $(\mathsf{m}_i, \mathsf{st}[\mathcal{P}_i])$

**distinguisher** $\mathcal{D}_i(x)$:
Return $x \oplus \neg i$

**adversary** $\mathcal{A}_2$:
Run $\mathcal{A}$ in the environment below

$\mathbf{Initialize}(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_2])$ from $\mathbf{Initialize}(\lambda)$
$(\mathsf{Rnd}[\mathcal{A}], \mathsf{Rnd}[\mathcal{K}]) \leftarrow \mathsf{Rnd}[\mathcal{A}_2]$
$\mathsf{st}[\mathcal{K}] \leftarrow (\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$
$\mathsf{List} \leftarrow []$
Return $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}])$

**query** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$:
$(\mathsf{m}, \mathsf{st}[\mathcal{K}]) \leftarrow_\$ \mathcal{K}(\mathsf{c}, \mathsf{PK}, \mathsf{R}, \mathsf{List}, \mathsf{st}[\mathcal{K}]; \mathsf{Rnd}[\mathcal{K}])$
Return $\mathsf{m}$

**query** $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$:
Query $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$ to get $\mathsf{c}$
$\mathsf{List} \leftarrow (\mathsf{c}, \mathsf{PK}^\star) : \mathsf{List}$
Return $\mathsf{c}$

**query** $\mathbf{Finalize}(d)$:
Query $\mathbf{Finalize}(d)$

Fig. 12: SPA2 ciphertext creator $\mathcal{A}_1$, plaintext creators $\mathcal{P}_0$ and $\mathcal{P}_1$, distinguishers $\mathcal{D}_0$ and $\mathcal{D}_1$, and IND-CPA adversary $\mathcal{A}_2$.

where games Real-SNM-SCCAx and Ideal-SNM-SCCAx are as shown in Figure 13.

## D Proof of Theorem 3: SPAx ∧ SNM-CPA ⇒ Non-Assisted SNM-SCCAx

*Proof.* We prove the theorem for $\mathsf{x} = 2$. Fix a definition of $\mathbf{SDecrypt}$, and let $\mathcal{A}$ be a Real-SNM-SCCA2 adversary against $\Pi$. Based on $\mathcal{A}$, we construct a SPA2 ciphertext creator $\mathcal{A}_1$, as shown on the top left in Figure 14, and let $\mathcal{K}$ be a plaintext extractor for $\mathcal{A}_1$. Using $\mathcal{A}$ and $\mathcal{K}$ we then construct a Real-SNM-CPA adversary $\mathcal{A}_2$ (also shown in Figure 14, right), and let $\mathcal{S}_2$ be a (non-assisted) simulator for it. We let the Ideal-SNM-SCCA2 simulator $\mathcal{S}$ to be the same as $\mathcal{S}_2$ and show that for a specific plaintext creator $\mathcal{P}$, and a distinguisher $\mathcal{D}$, both shown on the bottom left of Figure 14, we have:

$$\mathbf{Adv}^{\mathsf{snm\text{-}sccax}}_{\Pi,\mathsf{R},\mathcal{S},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\mathsf{spax}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{snm\text{-}cpa}}_{\Pi,\mathsf{R},\mathcal{S}_2}(\mathcal{A}_2).$$

**Claim D1.** *We have:*

$$\Pr\left[\mathsf{Real\text{-}SNM\text{-}SCCA2}^{\mathcal{A}}_{\Pi,\mathsf{R}}(\lambda) \Rightarrow \mathsf{T}\right] = \Pr\left[\mathsf{Dec\text{-}SPA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P},\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right]$$
$$\Pr\left[\mathsf{Real\text{-}SNM\text{-}CPA}^{\mathcal{A}_2}_{\Pi,\mathsf{R}}(\lambda) \Rightarrow \mathsf{T}\right] = \Pr\left[\mathsf{Ext\text{-}SPA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right]$$

Using this claim, which can be proven using standard arguments such as those presented in the previous section, the theorem follows:

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{snm\text{-}scca2}}_{\Pi,\mathsf{R},\mathcal{S},\mathcal{A}}(\lambda) &= \Pr\left[\mathsf{Real\text{-}SNM\text{-}SCCA2}^{\mathcal{A}}_{\Pi,\mathsf{R}}(\lambda) \Rightarrow \mathsf{T}\right] - \Pr\left[\mathsf{Ideal\text{-}SNM\text{-}SCCA2}^{\mathcal{S}}_{\Pi,\mathsf{R}}(\lambda) \Rightarrow \mathsf{T}\right] \\
&= \Pr\left[\mathsf{Dec\text{-}SPA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P},\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right] - \Pr\left[\mathsf{Ideal\text{-}SNM\text{-}CPA}^{\mathcal{S}_2}_{\Pi,\mathsf{R}}(\lambda) \Rightarrow \mathsf{T}\right] \\
&= \mathbf{Adv}^{\mathsf{spa2}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K},\mathcal{A}_1}(\lambda) + \Pr\left[\mathsf{Ext\text{-}SPA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right] - \Pr\left[\mathsf{Ideal\text{-}SNM\text{-}CPA}^{\mathcal{S}_2}_{\Pi,\mathsf{R}}(\lambda) \Rightarrow \mathsf{T}\right] \\
&= \mathbf{Adv}^{\mathsf{spa2}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K},\mathcal{A}_1}(\lambda) + \Pr\left[\mathsf{Real\text{-}SNM\text{-}CPA}^{\mathcal{A}_2}_{\Pi,\mathsf{R}}(\lambda) \Rightarrow \mathsf{T}\right] - \Pr\left[\mathsf{Ideal\text{-}SNM\text{-}CPA}^{\mathcal{S}_2}_{\Pi,\mathsf{R}}(\lambda) \Rightarrow \mathsf{T}\right] \\
&= \mathbf{Adv}^{\mathsf{spa2}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{snm\text{-}cpa}}_{\Pi,\mathsf{R},\mathcal{S}_2}(\mathcal{A}_2).
\end{aligned}$$

Game Real-SNM-SCCA$\mathsf{x}_{\Pi,\mathsf{R}}(\lambda)$

**procedure** $\mathbf{Initialize}(\lambda)$:
$\mathsf{I} \leftarrow_{\$} \mathsf{Setup}(1^{\lambda})$; $(\mathsf{SK}^{\star}, \mathsf{PK}^{\star}) \leftarrow_{\$} \mathsf{Gen}(\mathsf{I})$
$\mathsf{List} \leftarrow []$; $\mathsf{st}[\mathsf{V}] \leftarrow \mathsf{st}_0$
Return $(\mathsf{I}, \mathsf{PK}^{\star})$

**procedure** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R}')$:
Return $\mathbf{SDecrypt}_{\mathsf{U},\mathsf{V}}(\mathsf{c}, \mathsf{PK}, \mathsf{R}')$

**procedure** $\mathbf{Encrypt}(\mathrm{M}, \mathsf{st}_{\mathsf{R}})$:
$\mathsf{m} \leftarrow_{\$} \mathrm{M}()$; $\mathsf{c} \leftarrow_{\$} \mathsf{Enc}(\mathsf{m}, \mathsf{PK}^{\star})$
$\mathsf{List} \leftarrow (\mathsf{c}, \mathsf{PK}^{\star}) : \mathsf{List}$
Return $\mathsf{c}$

**procedure** $\mathbf{Finalize}(\mathbf{c}, \mathbf{PK}, \mathbf{R})$:
For $i$ from 1 to $\#\mathbf{c}$ do
  $\mathbf{m}[i] \leftarrow \mathbf{SDecrypt}(\mathbf{c}[i], \mathbf{PK}[i], \mathbf{R}[i])$
Return $\mathsf{R}(\mathsf{I}, \mathsf{m}, \mathbf{m}, \mathbf{c}, \mathsf{PK}^{\star}, \mathbf{PK}, \mathrm{M}, \mathsf{st}_{\mathsf{R}})$

Game Ideal-SNM-SCCA$\mathsf{x}_{\Pi,\mathsf{R}}(\lambda)$

**procedure** $\mathbf{Initialize}(\lambda)$:
$\mathsf{I} \leftarrow_{\$} \mathsf{Setup}(1^{\lambda})$; $(\mathsf{SK}^{\star}, \mathsf{PK}^{\star}) \leftarrow_{\$} \mathsf{Gen}(\mathsf{I})$
$\mathsf{st}[\mathsf{V}] \leftarrow \mathsf{st}_0$
Return $(\mathsf{I}, \mathsf{PK}^{\star})$

**procedure** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R}')$:
Return $\mathbf{SDecrypt}_{\mathsf{U},\mathsf{V}}(\mathsf{c}, \mathsf{PK}, \mathsf{R}')$

**procedure** $\mathbf{Finalize}(\mathbf{c}, \mathbf{PK}, \mathbf{R}, \mathrm{M}, \mathsf{st}_{\mathsf{R}})$:
For $i$ from 1 to $\#\mathbf{c}$ do
  $\mathbf{m}[i] \leftarrow \mathbf{SDecrypt}(\mathbf{c}[i], \mathbf{PK}[i], \mathbf{R}[i])$
$\mathsf{m} \leftarrow_{\$} \mathrm{M}()$
Return $\mathsf{R}(\mathsf{I}, \mathsf{m}, \mathbf{m}, \mathbf{c}, \mathsf{PK}^{\star}, \mathbf{PK}, \mathrm{M}, \mathsf{st}_{\mathsf{R}})$

Fig. 13: Games defining simulation-based complete non-malleability under strong chosen-ciphertext attacks. An adversary $\mathcal{A}$ is legitimate if: 1) It calls $\mathbf{Encrypt}$ once with a valid $\mathrm{M}$; 2) $\mathsf{R}'$ queried to $\mathbf{SDecrypt}$ is computable in polynomial time; if $\mathsf{x} = 0$ it does not call $\mathbf{SDecrypt}$; if $\mathsf{x} = 1$ it does not call $\mathbf{SDecrypt}$ after calling $\mathbf{Left}$-$\mathbf{Right}$; and if $\mathsf{x} = 2$ it does not call $\mathbf{SDecrypt}$ with a tuple in $\mathsf{List}$; and 3) It calls $\mathbf{Finalize}$ with a tuple such that all relations in $\mathbf{R}$ are computable in polynomial time and, for $i = 1, \ldots, \#\mathbf{c}$, the tuples $(\mathbf{c}[i], \mathbf{PK}[i])$ do not appear in $\mathsf{List}$. A *non-assisted* simulator $\mathcal{S}$ is legitimate if: 1) It calls $\mathbf{Finalize}$ with a valid $\mathrm{M}$; and 2) It does not call $\mathbf{SDecrypt}$. An *assisted* simulator is legitimate if: 1) It calls $\mathbf{Finalize}$ with a valid $\mathrm{M}$; 2) $\mathsf{R}'$ queried to $\mathbf{SDecrypt}$ is computable in polynomial time; and 3) If $\mathsf{x} = 0$ it does not call $\mathbf{SDecrypt}$.

□

REMARK. The proof of the theorem for the $\mathsf{x} = 1$ case is similar. Note that, although game SNM-SCCA1 involves an "implicit" adaptive decryption query at $\mathbf{Finalize}$, SKA1 secret key awareness is enough for the theorem to go through since game SNM-CPA also has this implicit adaptive decryption at its $\mathbf{Finalize}$ stage.

## E Proof of Theorem 4: SKAx ∧ IND-CCAx ⇒ IND-ICAx

*Proof.* We first prove the theorem for $\mathsf{x} = 1$. The proof technique is similar to that given for Theorem 2. Let $\mathcal{A}$ be an IND-ICA1 adversary against $\Pi$. Based on $\mathcal{A}$, we first construct an SKA1 public key creator $\mathcal{A}_1$, as shown on the top left of Figure 15, and let $\mathcal{K}$ be a secret key extractor for it. Based on $\mathcal{A}$ and $\mathcal{K}$, we then construct an IND-CCA1 adversary $\mathcal{A}_2$ against $\Pi$, as shown on the right of the same figure. Finally, we give distinguishers $\mathcal{D}_0$ and $\mathcal{D}_1$, included on the bottom left of Figure 15, and show that:

$$\mathbf{Adv}^{\mathsf{ind\text{-}ica1}}_{\Pi,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\mathsf{ska1}}_{\Pi,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ska1}}_{\Pi,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ind\text{-}cca1}}_{\Pi,\mathcal{A}_2}(\lambda).$$

**Claim E1.** *We have:*

$$\begin{aligned}
\Pr\left[\mathsf{IND\text{-}ICA1}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] &= \Pr\left[\mathsf{Inv\text{-}SKA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_1}(\lambda) \Rightarrow \mathsf{T}\right] \\
\Pr\left[\mathsf{IND\text{-}ICA1}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] &= 1 - \Pr\left[\mathsf{Inv\text{-}SKA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_0}(\lambda) \Rightarrow \mathsf{T}\right] \\
\Pr\left[\mathsf{IND\text{-}CCA1}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] &= \Pr\left[\mathsf{Ext\text{-}SKA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_1,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right] \\
\Pr\left[\mathsf{IND\text{-}CCA1}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] &= 1 - \Pr\left[\mathsf{Ext\text{-}SKA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_0,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right]
\end{aligned}$$

**adversary** $\mathcal{A}_1$:
Run $\mathcal{A}$ in the environment below

**Initialize**$(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_1])$ from **Initialize**$(\lambda)$
Return $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_1])$

**query SDecrypt**$(\mathsf{c}, \mathsf{PK}, \mathsf{R})$:
Query **SDecrypt**$(\mathsf{c}, \mathsf{PK}, \mathsf{R})$ to get $\mathsf{m}$
Return $\mathsf{m}$

**query Encrypt**$(\mathrm{M}, \mathsf{st}_\mathsf{R})$:
Query **Encrypt**$(\mathrm{M})$ to get $\mathsf{c}$.
Return $\mathsf{c}$

**query Finalize**$(\mathbf{c}, \mathbf{PK}, \mathbf{R})$:
For $i$ from 1 to $\#\mathbf{c}$ do
  $\mathbf{m}[i] \leftarrow$ **SDecrypt**$(\mathbf{c}[i], \mathbf{PK}[i], \mathbf{R}[i])$
$x \leftarrow (\mathsf{I}, \mathsf{m}, \mathbf{m}, \mathbf{c}, \mathsf{PK}^\star, \mathbf{PK}, \mathrm{M}, \mathsf{st}_\mathsf{R})$
Query **Finalize**$(x)$

**plaintext creator** $\mathcal{P}(\mathrm{M}, \mathsf{st}[\mathcal{P}])$:
$\mathsf{m} \leftarrow_\$ \mathrm{M}()$
Return $(\mathsf{m}, \mathsf{st}[\mathcal{P}])$

**distinguisher** $\mathcal{D}(x)$:
$(\mathsf{I}, \mathsf{m}, \mathbf{m}, \mathbf{c}, \mathsf{PK}^\star, \mathbf{PK}, \mathrm{M}, \mathsf{st}_\mathsf{R}) \leftarrow x$
Return $\mathsf{R}(\mathsf{I}, \mathsf{m}, \mathbf{m}, \mathbf{c}, \mathsf{PK}^\star, \mathbf{PK}, \mathrm{M}, \mathsf{st}_\mathsf{R})$

**adversary** $\mathcal{A}_2$:
Run $\mathcal{A}$ in the environment below

**Initialize**$(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_2])$ from **Initialize**$(\lambda)$
$(\mathsf{Rnd}[\mathcal{A}], \mathsf{Rnd}[\mathcal{K}]) \leftarrow \mathsf{Rnd}[\mathcal{A}_2]$
$\mathsf{st}[\mathcal{K}] \leftarrow (\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$
$\mathsf{List} \leftarrow []$
Return $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}])$

**query SDecrypt**$(\mathsf{c}, \mathsf{PK}, \mathsf{R})$:
$(\mathsf{m}, \mathsf{st}[\mathcal{K}]) \leftarrow_\$ \mathcal{K}(\mathsf{c}, \mathsf{PK}, \mathsf{R}, \mathsf{st}[\mathcal{K}]; \mathsf{Rnd}[\mathcal{K}])$
Return $\mathsf{m}$

**query Encrypt**$(\mathrm{M})$:
Query $\mathsf{Enc}(\mathsf{m}, \mathsf{PK}^\star)$ to get $\mathsf{c}$
$\mathsf{List} \leftarrow (\mathsf{c}, \mathsf{PK}^\star) : \mathsf{List}$
Return $\mathsf{c}$

**query Finalize**$(\mathbf{c}, \mathbf{PK}, \mathbf{R}, \mathsf{st}_\mathsf{R})$:
Query **Finalize**$(\mathbf{c}, \mathbf{PK}, \mathbf{R}, \mathsf{st}_\mathsf{R})$

Fig. 14: SPA2 ciphertext creator $\mathcal{A}_1$, plaintext creator $\mathcal{P}$, distinguisher $\mathcal{D}$, and Real-SNM-CPA adversary $\mathcal{A}_2$.

Using this claim, which can be proven using standard arguments such as those presented in Appendix B, the theorem follows:

$$
\begin{aligned}
\mathbf{Adv}^{\mathsf{ind\text{-}ica1}}_{\Pi,\mathcal{A}}(\lambda) &= \Pr\left[\mathsf{IND\text{-}ICA1}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] - \Pr\left[\mathsf{IND\text{-}ICA1}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] \\
&= \Pr\left[\mathsf{Inv\text{-}SKA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_1}(\lambda) \Rightarrow \mathsf{T}\right] - \left(1 - \Pr\left[\mathsf{Inv\text{-}SKA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_0}(\lambda) \Rightarrow \mathsf{T}\right]\right) \\
&= \Pr\left[\mathsf{Ext\text{-}SKA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_1,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right] - \left(1 - \Pr\left[\mathsf{Ext\text{-}SKA1}^{\mathcal{A}_1}_{\Pi,\mathcal{D}_0,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right]\right) \\
&\quad + \mathbf{Adv}^{\mathsf{ska1}}_{\Pi,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ska1}}_{\Pi,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) \\
&= \Pr\left[\mathsf{IND\text{-}CCA1}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] - \Pr\left[\mathsf{IND\text{-}CCA1}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] \\
&\quad + \mathbf{Adv}^{\mathsf{ska1}}_{\Pi,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ska1}}_{\Pi,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) \\
&= \mathbf{Adv}^{\mathsf{ind\text{-}cca1}}_{\Pi,\mathcal{A}_2}(\lambda) + \mathbf{Adv}^{\mathsf{ska1}}_{\Pi,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ska1}}_{\Pi,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda).
\end{aligned}
$$

We now prove the theorem for $\mathsf{x} = 2$. Let $\mathcal{A}$ be an IND-ICA2 adversary against $\Pi$. Based on $\mathcal{A}$, we first construct an SKA2 public key creator $\mathcal{A}_1$, as shown on the top left of Figure 16, and let $\mathcal{K}$ be a secret key extractor for it. We then construct an IND-CCA2 adversary $\mathcal{A}_2$ against $\Pi$, as shown on the right of the same figure. On the bottom left of Figure 16 we present plaintext creators $\mathcal{P}_0$ and $\mathcal{P}_1$ and distinguishers $\mathcal{D}_0$ and $\mathcal{D}_1$ such that

$$
\mathbf{Adv}^{\mathsf{ind\text{-}ica2}}_{\Pi,\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\mathsf{ska2}}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ska2}}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ind\text{-}cca2}}_{\Pi,\mathcal{A}_2}(\lambda).
$$

**adversary** $\mathcal{A}_1$:
Run $\mathcal{A}$ in the environment below

$\mathbf{Initialize}(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_1])$ from $\mathbf{Initialize}(\lambda)$
Return $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_1])$

**query** $\mathbf{Decrypt}(\mathsf{c})$:
Query $\mathbf{Decrypt}(\mathsf{c})$ to get $\mathsf{m}$
Return $\mathsf{m}$

**query** $\mathbf{Invert}(\mathsf{PK}, \mathsf{R})$:
Query $\mathbf{Invert}(\mathsf{PK}, \mathsf{R})$ to get $\mathsf{SK}$
Return $\mathsf{SK}$

**query** $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$:
$x \leftarrow (\mathsf{m}_0, \mathsf{m}_1, \mathsf{PK}^\star)$
Query $\mathbf{Finalize}(x)$

**distinguisher** $\mathcal{D}_i(x)$:
$(\mathsf{m}_0, \mathsf{m}_1, \mathsf{PK}^\star) \leftarrow x$
$\mathsf{c} \leftarrow_\$ \mathsf{Enc}(\mathsf{m}_i, \mathsf{PK}^\star)$
Resume $\mathcal{A}(\mathsf{c})$ to get $d$
Return $d \oplus \neg i$

**adversary** $\mathcal{A}_2$:
Run $\mathcal{A}$ in the environment below

$\mathbf{Initialize}(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_2])$ from $\mathbf{Initialize}(\lambda)$
$(\mathsf{Rnd}[\mathcal{A}], \mathsf{Rnd}[\mathcal{K}]) \leftarrow \mathsf{Rnd}[\mathcal{A}_2]$
$\mathsf{st}[\mathcal{K}] \leftarrow (\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$
$\mathsf{List} \leftarrow []$; $\mathsf{List}' \leftarrow []$
Return $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}])$

**query** $\mathbf{Decrypt}(\mathsf{c})$:
Query $\mathbf{Decrypt}(\mathsf{c})$ to get $\mathsf{m}$
$\mathsf{List}' \leftarrow \mathsf{m} : \mathsf{List}'$
Return $\mathsf{m}$

**query** $\mathbf{Invert}(\mathsf{PK}, \mathsf{R})$:
$(\mathsf{SK}, \mathsf{st}[\mathcal{K}]) \leftarrow_\$ \mathcal{K}(\mathsf{PK}, \mathsf{R}, \mathsf{List}, \mathsf{List}', \mathsf{st}[\mathcal{K}]; \mathsf{Rnd}[\mathcal{K}])$
Return $\mathsf{SK}$

**query** $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$:
Query $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$ to get $\mathsf{c}$
$\mathsf{List} \leftarrow (\mathsf{c}, \mathsf{PK}^\star) : \mathsf{List}$ Return $\mathsf{c}$

**query** $\mathbf{Finalize}(d)$:
Query $\mathbf{Finalize}(d)$

Fig. 15: SKA1 public key creator $\mathcal{A}_1$, distinguishers $\mathcal{D}_0$ and $\mathcal{D}_1$, and IND-CCA1 adversary $\mathcal{A}_2$.

**Claim E2.** *We have:*

$$\Pr\left[\mathsf{IND\text{-}ICA2}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] = \Pr\left[\mathsf{Inv\text{-}SKA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_1,\mathcal{D}_1}(\lambda) \Rightarrow \mathsf{T}\right]$$
$$\Pr\left[\mathsf{IND\text{-}ICA2}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] = 1 - \Pr\left[\mathsf{Inv\text{-}SKA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_0,\mathcal{D}_0}(\lambda) \Rightarrow \mathsf{T}\right]$$
$$\Pr\left[\mathsf{IND\text{-}CCA2}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] = \Pr\left[\mathsf{Ext\text{-}SKA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right]$$
$$\Pr\left[\mathsf{IND\text{-}CCA2}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] = 1 - \Pr\left[\mathsf{Ext\text{-}SKA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right]$$

Using this claim, which can be proven using standard arguments such as those presented in Appendix B, the theorem follows:

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{ind\text{-}ica2}}_{\Pi,\mathcal{A}}(\lambda) &= \Pr\left[\mathsf{IND\text{-}ICA2}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] - \Pr\left[\mathsf{IND\text{-}ICA2}^{\mathcal{A}}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] \\
&= \Pr\left[\mathsf{Inv\text{-}SKA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_1,\mathcal{D}_1}(\lambda) \Rightarrow \mathsf{T}\right] - \left(1 - \Pr\left[\mathsf{Inv\text{-}SKA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_0,\mathcal{D}_0}(\lambda) \Rightarrow \mathsf{T}\right]\right) \\
&= \Pr\left[\mathsf{Ext\text{-}SKA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right] - \left(1 - \Pr\left[\mathsf{Ext\text{-}SKA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right]\right) \\
&\quad + \mathbf{Adv}^{\mathsf{ska2}}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ska2}}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) \\
&= \Pr\left[\mathsf{IND\text{-}CCA2}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 1\right] - \Pr\left[\mathsf{IND\text{-}CCA2}^{\mathcal{A}_2}_{\Pi}(\lambda) \Rightarrow \mathsf{T} | b = 0\right] \\
&\quad + \mathbf{Adv}^{\mathsf{ska2}}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ska2}}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda) \\
&= \mathbf{Adv}^{\mathsf{ind\text{-}cca2}}_{\Pi,\mathcal{A}_2}(\lambda) + \mathbf{Adv}^{\mathsf{ska2}}_{\Pi,\mathcal{P}_0,\mathcal{D}_0,\mathcal{K},\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{ska2}}_{\Pi,\mathcal{P}_1,\mathcal{D}_1,\mathcal{K},\mathcal{A}_1}(\lambda).
\end{aligned}$$

□

## F Proof of Theorem 5: SKAx ∧ PAx(λ) ⇒ SPAx

*Proof.* We prove the theorem for $\mathsf{x} = 2$ (case $\mathsf{x} = 1$ being similar). Let $\mathcal{A}$ be a SPA2 ciphertext creator against $\Pi$ with respect to the $\mathbf{SDecrypt}$ procedure associated to $\mathbf{Invert}$ defined in Figure 5. To complete the proof, we will

**adversary** $\mathcal{A}_1$:
Run $\mathcal{A}$ in the environment below

$\mathbf{Initialize}(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_1])$ from $\mathbf{Initialize}(\lambda)$
Return $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_1])$

**query** $\mathbf{Decrypt}(\mathsf{c})$:
Query $\mathbf{Decrypt}(\mathsf{c})$ to get $\mathsf{m}$
Return $\mathsf{m}$

**query** $\mathbf{Invert}(\mathsf{PK}, \mathsf{R})$:
Query $\mathbf{Invert}(\mathsf{PK}, \mathsf{R})$ to get $\mathsf{SK}$
Return $\mathsf{SK}$

**query** $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$:
Query $\mathbf{Encrypt}(\mathsf{m}_0, \mathsf{m}_1)$ to get $\mathsf{c}$
Return $\mathsf{c}$

**query** $\mathbf{Finalize}(d)$:
Query $\mathbf{Finalize}(d)$

**plaintext creator** $\mathcal{P}_i(Q, \mathsf{st}[\mathcal{P}_i])$:
$(\mathsf{m}_0, \mathsf{m}_1) \leftarrow Q$
Return $(\mathsf{m}_i, \mathsf{st}[\mathcal{P}_i])$

**distinguisher** $\mathcal{D}_i(x)$:
Return $d \oplus \neg i$

**adversary** $\mathcal{A}_2$:
Run $\mathcal{A}$ in the environment below

$\mathbf{Initialize}(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_2])$ from $\mathbf{Initialize}(\lambda)$
$(\mathsf{Rnd}[\mathcal{A}], \mathsf{Rnd}[\mathcal{K}]) \leftarrow \mathsf{Rnd}[\mathcal{A}_2]$
$\mathsf{st}[\mathcal{K}] \leftarrow (\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$
$\mathsf{List} \leftarrow []$; $\mathsf{List}' \leftarrow []$
Return $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}])$

**query** $\mathbf{Decrypt}(\mathsf{c})$:
Query $\mathbf{Decrypt}(\mathsf{c})$ to get $\mathsf{m}$
$\mathsf{List}' \leftarrow \mathsf{m} : \mathsf{List}'$
Return $\mathsf{m}$

**query** $\mathbf{Invert}(\mathsf{PK}, \mathsf{R})$:
$(\mathsf{SK}, \mathsf{st}[\mathcal{K}]) \leftarrow_\$ \mathcal{K}(\mathsf{PK}, \mathsf{R}, \mathsf{List}, \mathsf{List}', \mathsf{st}[\mathcal{K}]; \mathsf{Rnd}[\mathcal{K}])$
Return $\mathsf{SK}$

**query** $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$:
Query $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$ to get $\mathsf{c}$
$\mathsf{List} \leftarrow (\mathsf{c}, \mathsf{PK}^\star) : \mathsf{List}$
Return $\mathsf{c}$

**query** $\mathbf{Finalize}(d)$:
Query $\mathbf{Finalize}(d)$

Fig. 16: SKA2 public key creator $\mathcal{A}_1$, plaintext creators $\mathcal{P}_0$ and $\mathcal{P}_1$, distinguishers $\mathcal{D}_0$ and $\mathcal{D}_1$, and IND-CCA2 adversary $\mathcal{A}_2$.

show that there exists an extractor for this adversary, and we will do so using the fact that the scheme is both secret key aware and plaintext aware. To simplify the proof, we present it as a sequence of games, with game $\mathsf{Game}_1$ being identical to Dec-SPA2.

Before introducing $\mathsf{Game}_2$ we observe that $\mathcal{A}$ can be used to construct a SKA2 public key creator that essentially runs $\mathcal{A}_1$ in the environment of $\mathsf{Game}_1$. This is shown on the left in Figure 19. We let $\mathcal{K}_1$ be the secret key extractor for this public key creator implied by the assumption that the scheme is secret key aware. Now, in game $\mathsf{Game}_2$, the adversary $\mathcal{A}$ is run in an environment that is identical to $\mathsf{Game}_1$ except that its decryption queries are answered using the hybrid procedure shown in Figure 17. The invert oracle is replaced by the secret key extractor, whose state is initialised with $(\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$ consistently with its operation in the SKA2 game.

**procedure** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$ of $\mathsf{Game}_2$
If $\mathsf{PK} = \mathsf{PK}^\star$
  Query $\mathbf{Decrypt}(\mathsf{c})$ to get $\mathsf{m}$
  $\mathsf{List}' \leftarrow \mathsf{c} : \mathsf{List}'$
  If $\mathsf{R}(\mathsf{m})$ Return $\mathsf{m}$ Else Return $\perp$
$\mathsf{R}'(\mathsf{SK}) := \mathsf{R}(\mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK}))$
$(\mathsf{SK}, \mathsf{st}[\mathcal{K}_1]) \leftarrow_\$ \mathcal{K}_1(\mathsf{PK}, \mathsf{R}', \mathsf{List}, \mathsf{List}', \mathsf{st}[\mathcal{K}_1]; \mathsf{Rnd}[\mathcal{K}_1])$
If $\mathsf{SK} = \perp$ Return $\perp$
$\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK})$
Return $\mathsf{m}$

Fig. 17: Procedure $\mathbf{SDecrypt}$ of game $\mathsf{Game}_2$.

We now observe that we can construct a PA2 ciphertext creator $\mathcal{A}_2$ (shown in Figure 19 on the right) using $\mathcal{A}$ and $\mathcal{K}_1$, that essentially runs $\mathcal{A}$ in the same environment as $\mathsf{Game}_2$. Note that the random coins of $\mathcal{A}_2$ include those of $\mathcal{A}$ and also $\mathcal{K}_1$. We let $\mathcal{K}_2$ be the plaintext extractor for it, which is implied by the assumption that the scheme is plaintext aware. Finally, we introduce $\mathsf{Game}_3$, where the adversary's decryption queries are now answered using a full strong plaintext extractor $\mathcal{K}$, constructed from $\mathcal{K}_1$ and $\mathcal{K}_2$, according to the procedure shown in Figure 18. Note that the states of the knowledge extractors must be initialised consistently with $(\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$ and $(\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}]||\mathsf{Rnd}[\mathcal{K}_1])$, respectively. We observe that $A$ is now being run in a game that is identical to Ext-SPA2, for the specific extractor $\mathcal{K}$.

**procedure** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$ of $\mathsf{Game}_3$
$(\mathsf{m}, \mathsf{st}[\mathcal{K}]) \leftarrow_\$ \mathcal{K}(\mathsf{c}, \mathsf{PK}, \mathsf{R}, \mathsf{List}, \mathsf{st}[\mathcal{K}]; \mathsf{Rnd}[\mathcal{K}])$
Return $\mathsf{m}$

**plaintext extractor** $\mathcal{K}(\mathsf{c}, \mathsf{PK}, \mathsf{R}, \mathsf{List}, \mathsf{st}[\mathcal{K}])$:
$(\mathsf{st}[\mathcal{K}_2], \mathsf{st}[\mathcal{K}_1], \mathsf{List}') \leftarrow \mathsf{st}[\mathcal{K}]$
If $\mathsf{PK} = \mathsf{PK}^\star$
  $(\mathsf{m}, \mathsf{st}[\mathcal{K}_2]) \leftarrow_\$ \mathcal{K}_2(\mathsf{c}, \mathsf{List}, \mathsf{st}[\mathcal{K}_2]; \mathsf{Rnd}[\mathcal{K}_2])$
  $\mathsf{List}' \leftarrow \mathsf{c} : \mathsf{List}'$
  If $\mathsf{R}(\mathsf{m})$ Return $\mathsf{m}$ Else Return $(\perp, (\mathsf{st}[\mathcal{K}_2], \mathsf{st}[\mathcal{K}_1], \mathsf{List}'))$
$\mathsf{R}'(\mathsf{SK}) := \mathsf{R}(\mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK}))$
$(\mathsf{SK}, \mathsf{st}[\mathcal{K}_1]) \leftarrow_\$ \mathcal{K}_1(\mathsf{PK}, \mathsf{R}', \mathsf{List}, \mathsf{List}', \mathsf{st}[\mathcal{K}_1]; \mathsf{Rnd}[\mathcal{K}_1])$
If $\mathsf{SK} = \perp$ Return $(\perp, (\mathsf{st}[\mathcal{K}_2], \mathsf{st}[\mathcal{K}_1], \mathsf{List}'))$
$\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK})$
Return $(\mathsf{m}, (\mathsf{st}[\mathcal{K}_2], \mathsf{st}[\mathcal{K}_1], \mathsf{List}'))$

Fig. 18: Procedure $\mathbf{SDecrypt}$ of $\mathsf{Game}_3$ using SPA2 plaintext extractor $\mathcal{K}$.

**Claim F1.** *We have:*

$$\begin{aligned}
\Pr\left[\mathsf{Inv\text{-}SKA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P},\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right] &= \Pr\left[\mathsf{Dec\text{-}SPA2}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right] \\
\Pr\left[\mathsf{Ext\text{-}SKA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}_1}(\lambda) \Rightarrow \mathsf{T}\right] &= \Pr\left[\mathsf{Game}_2{}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}_1}(\lambda) \Rightarrow \mathsf{T}\right] \\
\Pr\left[\mathsf{Dec\text{-}PA2}^{\mathcal{A}_2}_{\Pi,\mathcal{P},\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right] &= \Pr\left[\mathsf{Game}_2{}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}_1}(\lambda) \Rightarrow \mathsf{T}\right] \\
\Pr\left[\mathsf{Ext\text{-}PA2}^{\mathcal{A}_2}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}_2}(\lambda) \Rightarrow \mathsf{T}\right] &= \Pr\left[\mathsf{Ext\text{-}SPA2}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right]
\end{aligned}$$

Using this claim, which can be proven using standard arguments such as those presented in Appendix B, the theorem follows:

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{spax}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K},\mathcal{A}}(\lambda) &= \Pr\left[\mathsf{Dec\text{-}SPAx}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right] - \Pr\left[\mathsf{Ext\text{-}SPAx}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right] \\
&= \Pr\left[\mathsf{Dec\text{-}SPAx}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right] - \Pr\left[\mathsf{Game}_2{}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}_1}(\lambda) \Rightarrow \mathsf{T}\right] \\
&\quad + \Pr\left[\mathsf{Game}_2{}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}_1}(\lambda) \Rightarrow \mathsf{T}\right] - \Pr\left[\mathsf{Ext\text{-}SPAx}^{\mathcal{A}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right] \\
&= \Pr\left[\mathsf{Inv\text{-}SKA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P},\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right] - \Pr\left[\mathsf{Ext\text{-}SKA2}^{\mathcal{A}_1}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}_1}(\lambda) \Rightarrow \mathsf{T}\right] \\
&\quad + \Pr\left[\mathsf{Dec\text{-}PA2}^{\mathcal{A}_2}_{\Pi,\mathcal{P},\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right] - \Pr\left[\mathsf{Ext\text{-}PA2}^{\mathcal{A}_2}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}_2}(\lambda) \Rightarrow \mathsf{T}\right] \\
&= \mathbf{Adv}^{\mathsf{skax}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}_1,\mathcal{A}_1}(\lambda) + \mathbf{Adv}^{\mathsf{pax}}_{\Pi,\mathcal{P},\mathcal{D},\mathcal{K}_2,\mathcal{A}_2}(\lambda).
\end{aligned}$$

□

**adversary** $\mathcal{A}_1$:
Run $\mathcal{A}$ in the environment below

$\mathbf{Initialize}(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_1])$ from $\mathbf{Initialize}(\lambda)$
Return $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_1])$

**query** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$:
If $\mathsf{PK} = \mathsf{PK}^\star$
  Query $\mathbf{Decrypt}(\mathsf{c})$ to get $\mathsf{m}$
  If $\mathsf{R}(\mathsf{m})$ Return $\mathsf{m}$ Else Return $\perp$
$\mathsf{R}'(\mathsf{SK}) := \mathsf{R}(\mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK}))$
Query $\mathbf{Invert}(\mathsf{PK}, \mathsf{R}')$ to get $\mathsf{SK}$
If $\mathsf{SK} = \perp$ Return $\perp$
$\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK})$
Return $\mathsf{m}$

**query** $\mathbf{Encrypt}(Q)$:
Query $\mathbf{Encrypt}(Q)$ to get $\mathsf{c}$
Return $\mathsf{c}$

**query** $\mathbf{Finalize}(x)$:
Query $\mathbf{Finalize}(x)$

**adversary** $\mathcal{A}_2$:
Run $\mathcal{A}$ in the environment below

$\mathbf{Initialize}(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}_2])$ from $\mathbf{Initialize}(\lambda)$
$(\mathsf{Rnd}[\mathcal{A}], \mathsf{Rnd}[\mathcal{K}_1]) \leftarrow \mathsf{Rnd}[\mathcal{A}_2]$
$\mathsf{st}[\mathcal{K}_1] \leftarrow (\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$
Return $(\mathsf{I}, \mathsf{PK}^\star; \mathsf{Rnd}[\mathcal{A}])$

**query** $\mathbf{SDecrypt}(\mathsf{c}, \mathsf{PK}, \mathsf{R})$:
If $\mathsf{PK} = \mathsf{PK}^\star$
  Query $\mathbf{Decrypt}(\mathsf{c})$ to get $\mathsf{m}$
  $\mathsf{List}' \leftarrow \mathsf{c} : \mathsf{List}'$
  If $\mathsf{R}(\mathsf{m})$ Return $\mathsf{m}$ Else Return $\perp$
$\mathsf{R}'(\mathsf{SK}) := \mathsf{R}(\mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK}))$
$(\mathsf{SK}, \mathsf{st}[\mathcal{K}_1]) \leftarrow_{\$} \mathcal{K}_1(\mathsf{PK}, \mathsf{R}', \mathsf{List}, \mathsf{List}', \mathsf{st}[\mathcal{K}_1]; \mathsf{Rnd}[\mathcal{K}_1])$
If $\mathsf{SK} = \perp$ Return $\perp$ Else $\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK})$
Return $\mathsf{m}$

**query** $\mathbf{Encrypt}(Q)$:
Query $\mathbf{Encrypt}(Q)$ to get $\mathsf{c}$
Return $\mathsf{c}$

**query** $\mathbf{Finalize}(x)$:
Query $\mathbf{Finalize}(x)$

Fig. 19: SKA2 public key creator $\mathcal{A}_1$ and PA2 ciphertext creator $\mathcal{A}_2$.

## G Generic transformation to SKA2 in the random oracle model

**Theorem 6.** *Let $\Pi$ be an encryption scheme, and set $\Pi'$ to be the scheme obtained by applying the transformation in Figure 8 to $\Pi$. Suppose further that the size of the codomain of $\mathsf{H}$ is $\ell_{\mathsf{H}}$ and let $\max(\mathsf{SKSp})$ denote the maximum $|\mathsf{SKSp}(\mathsf{PK})|$ over all public keys, where*

$$\mathsf{SKSp}(\mathsf{PK}) = \{\mathsf{SK} : \mathsf{KeySp}(\mathsf{PK}, \mathsf{SK})\}.$$

*Then, in the random oracle model, we have:*

1. *For any* IND-CCAx *adversary $\mathcal{A}$ against $\Pi'$ there exists an* IND-CCAx *adversary $\mathcal{A}_1$ against $\Pi$ such that*

$$\mathbf{Adv}^{\mathsf{ind\text{-}ccax}}_{\Pi',\mathcal{A}}(\lambda) = \mathbf{Adv}^{\mathsf{ind\text{-}ccax}}_{\Pi,\mathcal{A}_1}(\lambda);$$

2. *Fix the definition of the* $\mathbf{Invert}$ *oracle based on the natural validity criteria that accepts a key pair if and only if it could have been honestly generated. Then, there is a (universal) secret key extractor $\mathcal{K}$ such that for any* SKA2 *public key creator $\mathcal{A}$ against $\Pi'$ making at most $Q_{\mathsf{Inv}}$ public key inversion queries, for any probabilistic polynomial-time plaintext creator $\mathcal{P}$ and any unbounded $\mathcal{D}$ we have:*

$$\mathbf{Adv}^{\mathsf{ska2}}_{\Pi',\mathcal{P},\mathcal{D},\mathcal{K},\mathcal{A}}(\lambda) \leq Q_{\mathsf{Inv}} \left(1 - \left(1 - \frac{\max(\mathsf{KeySp})}{2^{\ell_{\mathsf{H}}}}\right)^{\max(\mathsf{KeySp})} + \frac{\max(\mathsf{SKSp})}{2^{\ell_{\mathsf{H}}}}\right).$$

*Proof.* 1) Given an IND-CCAx adversary $\mathcal{A}$, we construct an IND-CCAx adversary $\mathcal{A}_1$ as shown in Figure 20. Adversary $\mathcal{A}_1$ perfectly simulates the environment for $\mathcal{A}$ unless it queries the hash function on a key-pair such that the public key matches the challenge public key and the secret key corresponds to $\mathsf{PK}^\star$. Since key-pairs can be checked for validity, this event is detectable, and $\mathcal{A}_1$ can easily win its game.

**adversary** $\mathcal{A}_1$:
Run $\mathcal{A}$ in the environment below

$\mathbf{Initialize}(\lambda)$:
Get $(\mathsf{I}, \mathsf{PK}^\star)$ from $\mathbf{Initialize}(\lambda)$
$h^\star \leftarrow_\$ \{0,1\}^{\ell_\mathsf{H}}$
Return $(\mathsf{I}, (\mathsf{PK}^\star, h^\star))$

**query** $\mathbf{Decrypt}(\mathsf{c})$:
Query $\mathbf{Decrypt}(\mathsf{c})$ to get $\mathsf{m}$
Return $\mathsf{m}$

**query** $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$:
Query $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$ to get $\mathsf{c}$
Return $\mathsf{c}$

**query** $\mathbf{Finalize}(d)$:
Query $\mathbf{Finalize}(d)$

**query** $\mathsf{H}(\mathsf{SK}, \mathsf{PK})$:
If $\mathsf{PK} = \mathsf{PK}^\star \wedge \mathsf{KeySp}(\mathsf{SK}, \mathsf{PK})$
  If $\mathbf{Left\text{-}Right}$ not called
    Query $\mathbf{Left\text{-}Right}(\mathsf{m}_0, \mathsf{m}_1)$ to get $\mathsf{c}$
  $\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{c}, \mathsf{SK}, \mathsf{PK})$
  If $\mathsf{m} = \mathsf{m}_0$ query $\mathbf{Finalize}(0)$
  Query $\mathbf{Finalize}(1)$
Return $\mathsf{RO}(\mathsf{SK}, \mathsf{PK})$

Fig. 20: IND-CCAx adversary $\mathcal{A}_1$ against $\Pi$ based on an IND-CCAx adversary $\mathcal{A}$ against $\Pi'$. Here $\mathsf{m}_0 \neq \mathsf{m}_1$.

2) We construct the random oracle model secret key extractor $\mathcal{K}$ as shown in Figure 21. We first observe that $\ell_\mathsf{H}$ and $\max(\mathsf{KeySp})$ guarantee a lower bound on the probability that the transformed encryption scheme will have a unique secret key for each valid public key:

$$1 - \epsilon = \left(1 - \frac{\max(\mathsf{KeySp})}{2^{\ell_\mathsf{H}}}\right)^{\max(\mathsf{KeySp})}.$$

Indeed, the collision resistance property of the hash function will guarantee that no two valid secret keys in the original scheme will map to the same hash value and hence lead to different public keys in the new scheme with high probability. This means that the distribution of the secret keys returned by the secret key extractor, for each query where the hash list provides a candidate secret key $\mathsf{SK}$, will be incorrect with probability at most $\epsilon$.

Furthermore, the simulation may be inconsistent in the cases when a valid public key $(\mathsf{PK}, h)$ is queried to the $\mathbf{Invert}$ oracle but the random oracle $\mathsf{H}$ has not been queried on a valid corresponding $(\mathsf{SK}, \mathsf{PK})$ and $h$ is the correct hash value. It is easy to see that in this case $\perp$ is a invalid answer with probability at most $\max(\mathsf{SKSp})/2^{\ell_\mathsf{H}}$. Since there are $Q_{\mathsf{Inv}}$ queries to $\mathbf{Invert}$ we obtain the bound

$$Q_{\mathsf{Inv}}\left(1 - \left(1 - \frac{\max(\mathsf{KeySp})}{2^{\ell_\mathsf{H}}}\right)^{\max(\mathsf{KeySp})} + \frac{\max(\mathsf{SKSp})}{2^{\ell_\mathsf{H}}}\right).$$

**procedure** $\mathbf{Invert}(\mathsf{PK}, \mathsf{R})$:
$(\mathsf{PK}', h) \leftarrow \mathsf{PK}$
$\mathsf{WitSp} \leftarrow \{(\mathsf{SK}, r) : (\mathsf{PK}', \mathsf{SK}) \leftarrow \mathsf{Gen}(\mathsf{I}; r) \ \wedge \ h = \mathsf{H}(\mathsf{PK}', \mathsf{SK})\}$
$(\mathsf{SK}, r) \leftarrow_\$ \{(\mathsf{SK}, r) \in \mathsf{WitSp} : \mathsf{R}(\mathsf{SK})\}$
Return $\mathsf{SK}$

$\mathcal{K}(\mathsf{PK}, R, \mathsf{List}, \mathsf{List}', \mathsf{HList}, \mathsf{st}[\mathcal{K}])$:
$(\mathsf{PK}', h) \leftarrow \mathsf{PK}$
If $(\mathsf{SK}', \mathsf{PK}', h) \in \mathsf{HList} \wedge \mathsf{KeySp}(\mathsf{SK}', \mathsf{PK}')$
  Then if $\mathsf{R}(\mathsf{SK}')$ then Return $(\mathsf{SK}', \mathsf{st}[\mathcal{K}])$
Return $(\perp, \mathsf{st}[\mathcal{K}])$

Fig. 21: Inversion oracle and matching key extractor $\mathcal{K}$ in the random oracle model.

□

## H A concrete RSA-based scheme

We first state the computational intractability assumption that underlies the standard notion of security of our scheme. Our assumption differs from the standard RSA assumptions in two respects: 1) we fix the encryption exponent to a specific value; and 2) we grant adversary access to an inversion oracle (when $\mathsf{x} = 1$). Details of this

assumption can be found in Figure 22. We define the RSAx advantage of an adversary against group scheme $\mathcal{G}_e$ with respect to knowledge extractor $\mathcal{K}$ as:

$$\mathbf{Adv}^{\mathsf{rsax}}_{\mathcal{G}_e,\mathcal{A}}(\lambda) := \Pr\left[\mathsf{RSAx}^{\mathcal{A}}_{\mathcal{G}_e}(\lambda) \Rightarrow \mathsf{T}\right].$$

Game $\mathsf{RSAx}_{\mathcal{G}_e}(\lambda)$

**procedure Initialize**$(\lambda)$:
$(P^\star, Q^\star) \leftarrow_\$ \mathcal{G}_e(1^\lambda)$ s.t. $\gcd(e, P^\star(P^\star-1)(Q^\star-1)) = 1$
$N^\star \leftarrow P^{\star 2}Q^\star$; $d \leftarrow 1/e \pmod{\varphi(N^\star)}$
Return $N^\star$

**procedure Root**$(y)$:
$t \leftarrow y^d \pmod{N^*}$
$(x, x') \leftarrow t$
Return $x$

**procedure Challenge**$()$:
$x^\star \leftarrow_\$ \mathbb{Z}_{N^\star}$
$y^\star \leftarrow x^{\star e} \pmod{N^\star}$
Return $y^\star$

**procedure Finalize**$(x)$:
Return $(x = x^\star)$

Fig. 22: Game defining fixed exponent RSA assumption. An adversary $\mathcal{A}$ is legitimate if: 1) If $\mathsf{x} = 0$ it never queries **Root**.; and 2) It never queries **Root** after **Challenge**. **Root** returns the first $\ell$ bits of $t$, i.e. $|x| = \ell$.

Based on the above two assumptions, we construct a public-key encryption scheme as shown in Figure 23.

Scheme $\Pi_{\mathcal{G}_e}$

**procedure** $\mathsf{Setup}(1^\lambda)$:
Return $1^\lambda$

**procedure** $\mathsf{Gen}(1^\lambda)$:
$(P^\star, Q^\star) \leftarrow_\$ \mathcal{G}_e(1^\lambda)$
$N^\star \leftarrow P^{\star 2}Q^\star$
$d \leftarrow 1/e \pmod{\varphi(N^\star)}$
Return $(d, N^\star)$

**procedure** $\mathsf{Enc}(\mathsf{m}, \mathsf{PK})$:
$r \leftarrow_\$ \{0,1\}^{|N^\star|-\ell-1}$
$\mathsf{c} \leftarrow (\mathsf{m}||r)^e \pmod{N^\star}$
Return $\mathsf{c}$

**procedure** $\mathsf{Dec}(\mathsf{c}, \mathsf{PK}, \mathsf{SK})$:
$t \leftarrow \mathsf{c}^d \pmod{N^\star}$
$(\mathsf{m}, r) \leftarrow t$
Return $\mathsf{m}$

Fig. 23: Public-key encryption scheme based on RSA. The message space of this scheme consists of bit-strings of length at most $\ell$, with $\ell$ a logarithmic function in the security parameter.

**Theorem 7.** *Under the* RSAx *assumption, for* $\mathsf{x} = 0, 1$*, the scheme in Figure 23 is* IND-CCAx *secure.*

The proof of the above theorem for $\mathsf{x} = 0$ and standard RSA modulus can be found in [Gol04]. This proof needs to be adapted to $P^2Q$ type modulus, fixed exponent in RSA permutation, and where a **Root** oracle is present. We omit the details as they are not relevant to the main topic of this paper.

**Theorem 8** (KFAx $\Rightarrow$ SKAx **for** $\mathsf{x} = 0, 1$)**.** *Let $\mathcal{A}$ be an* SKAx *adversary against the scheme in Figure 23. Based on $\mathcal{A}$ we construct a* KFAx *adversary $\mathcal{A}_1$ and let $\mathcal{K}_1$ be a factorisation extractor for it. Then using $\mathcal{K}_1$ we give a secret key extractor $\mathcal{K}$ such that for any (even unbounded) $\mathcal{D}$ we have:*

$$\mathbf{Adv}^{\mathsf{skax}}_{\Pi_{\mathcal{G}_e},\mathcal{D},\mathcal{K},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\mathsf{kfax}}_{\mathcal{G}_e,\mathcal{K}_1,\ell,\mathcal{A}_1}(\lambda).$$

*Proof.* We prove the theorem for $\mathsf{x} = 1$. Let $\mathcal{A}$ be a SKA1 public key creator[22]. Based on $\mathcal{A}$ we construct a KFA1 adversary $\mathcal{A}_1$ as shown in Figure 24. We then let $\mathcal{K}_1$ be a factorisation extractor for it. Based on $\mathcal{K}_1$ we construct a secret key extractor $\mathcal{K}$ for the scheme as shown in Figure 25.

We now define two games as shown in Figure 26.

[22] We omit the relation from the **Invert** procedure as there is exactly one secret key corresponding to each public key.

**query** $\mathbf{Initialize}(\lambda)$:
Get $(N^\star, \mathsf{Rnd}[\mathcal{A}_1])$ from $\mathbf{Initialize}$
$\mathsf{Rnd}[\mathcal{A}] \leftarrow \mathsf{Rnd}[\mathcal{A}_1]$
Return $(N^\star, \mathsf{Rnd}[\mathcal{A}])$

**query** $\mathbf{Decrypt}(\mathsf{c})$:
$y \leftarrow \mathsf{c}$
Query $\mathbf{Root}(y)$ to get $x$
$\mathsf{m} \leftarrow x$
Return $\mathsf{m}$

**query** $\mathbf{Invert}(\mathsf{PK})$:
$N \leftarrow \mathsf{PK}$
Query $\mathbf{Factor}(N)$ to get $(P, Q)$
If $P^2Q = N \wedge P \neq 1$ Return $1/e \pmod{P(P-1)(Q-1)}$
Return $\perp$

**query** $\mathbf{Finalize}(x)$:
Query $\mathbf{Finalize}()$

Fig. 24: KFA1 adversary $\mathcal{A}_1$ based on SKA1 adversary $\mathcal{A}$.

**extractor** $\mathcal{K}(\mathsf{PK}, \mathsf{List}', \mathsf{st}[\mathcal{K}]; \mathsf{Rnd}[\mathcal{K}])$:
$\mathsf{Rnd}[\mathcal{K}_1] \leftarrow \mathsf{Rnd}[\mathcal{K}]$; $\mathsf{st}[\mathcal{K}_1] \leftarrow \mathsf{st}[\mathcal{K}]$; $N \leftarrow \mathsf{PK}$
$((P, Q), \mathsf{st}[\mathcal{K}_1]) \leftarrow_\$ \mathcal{K}_1(N, \mathsf{List}', \mathsf{st}[\mathcal{K}_1]; \mathsf{Rnd}[\mathcal{K}])$
If $P^2Q = N \wedge P \neq 1$ Return $(1/e \pmod{P(P-1)(Q-1)}, \mathsf{st}[\mathcal{K}_1])$
Return $(\perp, \mathsf{st}[\mathcal{K}_1])$

Fig. 25: SKA1 extractor $\mathcal{K}$ based on KFA1 extractor $\mathcal{K}_1$.

$\mathsf{Game}_0$ is a restatement of the Inv-SKA1 game with respect to our scheme and hence:

$$\Pr\left[\mathsf{Inv\text{-}SKA1}^{\mathcal{A}}_{\Pi_{\mathcal{G}_e},\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right] = \Pr\left[\mathsf{Game}_0{}^{\mathcal{A}}_{\mathcal{G}_e,\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right].$$

Also by inspecting the codes of $\mathcal{K}$ and $\mathcal{K}_1$ and their respective inputs we see that:

$$\Pr\left[\mathsf{Ext\text{-}SKA1}^{\mathcal{A}}_{\Pi_{\mathcal{G}_e},\mathcal{D},\mathcal{K}}(\lambda) \Rightarrow \mathsf{T}\right] = \Pr\left[\mathsf{Game}_1{}^{\mathcal{A}}_{\mathcal{G}_e,\mathcal{D},\mathcal{K}_1}(\lambda) \Rightarrow \mathsf{T}\right].$$

Furthermore for any (even unbounded) $\mathcal{D}$:

$$\Pr\left[\mathsf{Game}_0{}^{\mathcal{A}}_{\mathcal{G}_e,\mathcal{D}}(\lambda) \Rightarrow \mathsf{T}\right] - \Pr\left[\mathsf{Game}_1{}^{\mathcal{A}}_{\mathcal{G}_e,\mathcal{D},\mathcal{K}_1}(\lambda) \Rightarrow \mathsf{T}\right] \leq \Pr[\mathsf{BD}],$$

where BD is the event that the $\mathbf{Invert}$ procedure in $\mathsf{Game}_1$ fails to match that of $\mathsf{Game}_0$. This is the only event which prevents the two games from being identical. However, BD is to identical Flag being set to T when $\mathcal{K}_1$ is used to extract factorisations of integers produced by the adversary $\mathcal{A}_1$. Hence:

$$\Pr[\mathsf{BD}] = \Pr\left[\mathsf{KFA1}^{\mathcal{A}_1}_{\mathcal{G}_e,\mathcal{K}_1}(\lambda) \Rightarrow \mathsf{T}\right].$$

We conclude that:

$$\mathbf{Adv}^{\mathsf{ska1}}_{\Pi_{\mathcal{G}_e},\mathcal{D},\mathcal{K},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\mathsf{kfa1}}_{\mathcal{G}_e,\mathcal{K}_1,\mathcal{A}_1}(\lambda).$$

□

Together with Theorems 4 and 1, it follows that above scheme is IND-SCCA1 secure.

$\mathsf{Game}_0(\lambda)$

**procedure Initialize**$(\lambda)$:
$\mathsf{I} \leftarrow \epsilon$; $(P^\star, Q^\star) \leftarrow_\$ \mathcal{G}_e(1^\lambda)$;
$N^\star \leftarrow P^{\star 2} Q^\star$; $d \leftarrow 1/e \pmod{\varphi(N^\star)}$
$(\mathsf{SK}^\star, \mathsf{PK}^\star) \leftarrow (d, N^\star)$
Choose coins $\mathsf{Rnd}[\mathcal{A}]$ for $\mathcal{A}$
$\mathsf{st}[\mathcal{P}] \leftarrow \epsilon$; $\mathsf{List}' \leftarrow []$; $\mathsf{st}[\mathsf{V}] \leftarrow \mathsf{st}_0$
Return $(\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$

**procedure Finalize**$(x)$:
Return $\mathcal{D}(x)$

**procedure Invert**$(\mathsf{PK})$:
$N \leftarrow \mathsf{PK}$
If $\exists P, Q$ s.t. $P^2 Q = N \wedge P \neq 1$
  Return $1/e \pmod{P(P-1)(Q-1)}$
Return $\perp$

**procedure Decrypt**$(\mathsf{c})$:
$t \leftarrow \mathsf{c}^d \pmod{N^\star}$
$(\mathsf{m}, r) \leftarrow t$
$\mathsf{List}' \leftarrow (\mathsf{m}, \mathsf{c}) : \mathsf{List}'$
Return $\mathsf{m}$

$\mathsf{Game}_1(\lambda)$

**procedure Initialize**$(\lambda)$:
$\mathsf{I} \leftarrow \epsilon$; $(P^\star, Q^\star) \leftarrow_\$ \mathcal{G}_e(1^\lambda)$;
$N^\star \leftarrow P^{\star 2} Q^\star$; $d \leftarrow 1/e \pmod{\varphi(N^\star)}$
$(\mathsf{SK}^\star, \mathsf{PK}^\star) \leftarrow (d, N^\star)$
Choose coins $\mathsf{Rnd}[\mathcal{A}]$ for $\mathcal{A}$
$\mathsf{st}[\mathcal{P}] \leftarrow \epsilon$; $\mathsf{List}' \leftarrow []$; $\mathsf{st}[\mathsf{V}] \leftarrow \mathsf{st}_0$
Return $(\mathsf{I}, \mathsf{PK}^\star, \mathsf{Rnd}[\mathcal{A}])$

**procedure Finalize**$(x)$:
Return $\mathcal{D}(x)$

**procedure Invert**$(\mathsf{PK})$:
$N \leftarrow \mathsf{PK}$
$((P, Q), \mathsf{st}[\mathcal{K}_1]) \leftarrow_\$ \mathcal{K}_1(N, \mathsf{List}', \mathsf{st}[\mathcal{K}_1])$
If $P^2 Q = N \wedge P \neq 1$ Return $(1/e \pmod{P(P-1)(Q-1)})$
Return $\perp$

**procedure Decrypt**$(\mathsf{c})$:
$t \leftarrow \mathsf{c}^d \pmod{N^\star}$
$(\mathsf{m}, r) \leftarrow t$
$\mathsf{List}' \leftarrow (\mathsf{m}, \mathsf{c}) : \mathsf{List}'$
Return $\mathsf{m}$

Fig. 26: Games used in the proof of Theorem 8