2017 IEEE 30th Computer Security Foundations Symposium

UC-Secure Non-Interactive Public-Key Encryption

Jan Camenisch*, Anja Lehmann*, Gregory Neven*, Kai Samelin*§
*IBM Research – Zurich, Rüschlikon, Switzerland
{jca,anj,nev,ksa}@zurich.ibm.com
§TU Darmstadt, Darmstadt, Germany

***Abstract*—The universal composability (UC) framework enables the modular design of cryptographic protocols by allowing arbitrary compositions of lower-level building blocks. Public-key encryption is unarguably a very important such building block. However, so far no UC-functionality exists that offers non-interactive encryption necessary for modular protocol construction. We provide an ideal functionality for non-committing encryption (i.e., public-key encryption secure against adaptive corruptions) with locally generated, and therefore non-interactive, ciphertexts. As a sanity check, we also provide a property-based security notion that we prove to be equivalent to the UC notion. We then show that the encryption scheme of Camenisch et al. (SCN '16) based on trapdoor permutations securely implements our notion in the random-oracle model without assuming secure erasures. This is the best one can hope to achieve as standard-model constructions do not exist due to the uninstantiability of round-optimal adaptively secure message transfer in the standard model (Nielsen, Crypto '02). We illustrate the modular reusability of our functionality by constructing the first non-interactive signcryption scheme secure against adaptive corruptions without secure erasures in the UC framework.**

I. Introduction

The universal composability (UC) framework [13] enables the modular design of cryptographic protocols by analyzing the security of composed protocols in a so-called "hybrid" model, where subprotocols are replaced by their ideal functionalities, thereby eliminating the need for explicit reductions from the security of the individual building blocks in the overall security proof. Faithful to its name, the UC framework guarantees that any secure instantiation of the subprotocols yields a secure instantiation of the composed protocol.

A wide variety of UC-secure cryptographic primitives have appeared in the literature. Public-key encryption is an important such primitive that may not have received the attention that it deserves in the UC framework, especially given its widespread use to build cryptographic protocols. The original UC paper by Canetti [13] specifies an ideal functionality $\mathcal{F}_{\mathsf{PKE}}$ for public-key encryption and proves it equivalent to the property-based notion of indistinguishability against chosen-ciphertext attack (IND-CCA2). However, this result only holds for non-adaptive adversaries, that is, where the set of corrupted parties is fixed a priori for each protocol instance.

a) Non-Committing Encryption: Security against adaptive corruptions is obviously the more realistic notion for practical applications, but is notoriously difficult to achieve for public-key encryption because of the so-called *selective decommitment problem* [22], [34]. In a nutshell, the problem is the following. As long as both sender and receiver are honest, the simulator communicates "dummy" ciphertext values without knowing the actual plaintext. When later the receiver gets corrupted, the simulator must provide the adversary with a decryption key that makes these dummy ciphertexts decrypt to the correct messages, which are only handed to the simulator at the moment of corruption. Similarly, when the sender is corrupted, the simulator must provide encryption randomness that turns the messages into the dummy ciphertexts. One can easily solve the latter problem by securely erasing the randomness after encryption. However, secure erasures do not help to provide correct decryption keys when the receiver gets corrupted, and security cannot be proven. This is an unsatisfactory situation, as already noted in prior work [7], [23], [24], [36], [37].

Adaptively secure public-key encryption is therefore often referred to as *non-committing encryption* (NCE). Nielsen [45] showed that NCE cannot be achieved in the standard model by non-interactive protocols, i.e., protocols sending only one message per encryption. Canetti, Halevi, and Katz [16] circumvent Nielsen's impossibility result by periodically changing decryption keys and requiring a small amount of interaction between senders, namely, the current time period that has to be communicated.

Our main concern is that such interactive, i.e., non-local, constructions are not suitable for all use cases. Examples include scenarios where the sender and receiver are not on-line simultaneously, as is for instance the case for encrypted email which are typically stored on an IMAP server and are retrieved by the receiver at any time, possibly when the sender is no longer available. Indeed, many real-world communications are "fire-and-forget," where additional communication is impossible or too expensive, e.g., for UDP, sensor networks, etc.

Security in the random-oracle model [5] can be a reasonable price to pay for adaptive non-interactive encryption, especially when other building blocks in a composed protocol rely on random oracles already. Nielsen [45] provided a non-interactive construction in the random-oracle model, but presented it as a secure message transmission (SMT) protocol, which lets a sender send a secure message to a receiver but does not provide the environment any actual ciphertexts, which is the same as in the work done by Choi et al. [20]. This is problematic when a higher-level protocol needs to perform further operations on ciphertexts, such as signing, hashing, or re-encrypting ciphertexts. Another problem is that Nielsen's

DOI 10.1109/CSF.2017.14

construction assumes authenticated channels between the sender and receiver. When the underlying encryption scheme is used without authenticated channels, the scheme becomes malleable and thus does not satisfy the security properties that one expects.

b) Our Contributions: We present a new non-committing encryption functionality $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ that gives the encrypting party access to actual ciphertexts, much like Canetti's $\mathcal{F}_{\mathsf{PKE}}$ and Canetti-Halevi-Katz' $\mathcal{F}_{\mathsf{AFSE}}$, but unlike Nielsen's $\mathcal{F}_{\mathsf{SMT}}$. We also present a new property-based notion FULL-SIM that we prove to be equivalent to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and to be strictly stronger than most existing notions [30], [11], but incomparable to Canetti et al.'s notion [16]. Apart from acting as a sanity check for our UC functionality, our property-based notion might also be easier to prove the security of new schemes. We then consider the recently proposed encryption scheme by Camenisch et al. [11] (the CLNS scheme) based on trapdoor one-way permutations, which is a minor modification of Bellare-Rogaway's CCA-secure encryption scheme [5] as well as Nielsen's SMT scheme [45]. We prove that this scheme satisfies FULL-SIM, and hence securely realizes $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ against adaptive adversaries, in the random-oracle model. We do not require secure erasures, which is important in practice because realizing secure erasures is virtually impossible on modern hardware and software, as intermediate results may be copied or moved across the RAM, swap partitions, and SSD memory blocks.

Therefore, in scenarios where interactive encryption is simply not an option, we believe that our functionality $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and our random-oracle construction can be conveniently used as a building block to construct higher-level protocols and being able to analyze their security. In particular, our functionality allows, for the first time, to analyze adaptively-secure UC hybrid protocols without secure erasures which require non-interactive encryption. We demonstrate this with the example of signcryption, showing that the generic encrypt-then-sign construction from the property-based world also works in UC, yielding the first adaptively UC-secure signcryption scheme without erasures.

c) On Programmable Random Oracles: Because any realization of our $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ functionality immediately gives rise to a non-interactive secure message transmission protocol, Nielsen's impossibility result extends to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, meaning that $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ cannot be securely instantiated in the standard model. Moreover, Canetti, Jain, and Scafuro [18] pointed out that a globally accessible and programmable random oracle (or common reference string, for that matter) cannot be instantiated with a single hash function.

Recommending a provably uninstantiable scheme for use in cryptographic protocols is, of course, somewhat controversial. Most other examples of uninstantiable random-oracle schemes [15], [27] are contrived constructions that would never be considered for real-world use. The quite natural construction of the CLNS scheme clearly does not fall in this category. Rather, we would compare the CLNS scheme to highly practical random-oracle schemes, such as RSA-OAEP encryption [6] or Schnorr signatures [48], that are widely used in practice in spite of strong indications that they may not be securely instantiable [8], [47].

Stronger even, RSA-OAEP and Schnorr signatures are used in practice, in spite of the availability of reasonably efficient alternatives that are provably secure in the standard model. However, for non-interactive encryption, there is simply no choice: Nielsen's result shows that standard-model alternatives do not exist, even if one were willing to sacrifice on efficiency.

Security in the random-oracle model, whether programmable or not, is still meaningful in the sense that it does protect against "generic" adversaries that treat the hash function as a black box. At the very least, the random-oracle model provides some way to rigorously analyze the security of a scheme and exclude serious design flaws: a proof in the random-oracle model is obviously still highly preferable over no security proof at all. Moreover, while our main result focuses on security against adaptive corruptions in the programmable random-oracle model, it is useful to observe that the CLNS scheme remains secure against non-adaptive corruptions (i.e., IND-CCA2 secure) in the *non*-programmable random-oracle model [5], which can be instantiated with a global random oracle [18].

d) Related Work: The first non-committing encryption schemes [3], [14] were only able to encrypt single bits and require new public keys to be distributed for each encrypted bit. Unsurprisingly, in the light of Nielsen's impossibility result [45], most "two-sided" NCE schemes, i.e., where both sender and receiver can be corrupted, are interactive [2], [3], [10], [14], [16], [20], [21], [25], [28], [29], [31], [32], [42], [45], [51], meaning that either encryption and decryption cannot be done locally without additional communication, there is an a-priori upper bound on the number of ciphertexts created before a new key pair has to be generated, or there is no ciphertext at all.

Nielsen [45] and Camenisch et al. [11] present non-interactive schemes in the random-oracle model without this a priori bound. However, as discussed earlier, the former is presented as an SMT protocol and therefore not easily reusable as a UC building block; also, it implicitly assumes authenticated channels between parties. The latter scheme is the same as the construction in our paper, but is proven secure under a property-based (i.e., non-UC) definition and assumes secure erasures in the analysis. We provide a UC analysis of their scheme without assuming secure erasures.

Single-sided definitions of NCE have also been proposed, where only the sender *or* the receiver can be corrupted [4], [7], [23], [24], [30], [35], [36], [37], [38], but not both at the same time. On the upside of these definitions is that there are realizations in the standard, i.e., non-random-oracle model, while some still resort to idealized models [33]. More importantly, these constructions are not proven secure if both sides can be corrupted.

II. Preliminaries

a) Notation: Let $\tau \in \mathbb{N}$ be the security parameter and 1^τ the string of τ ones. All algorithms implicitly receive 1^τ as their first input parameter. If S is a set, $a \xleftarrow{r} S$ denotes that

a is assigned a uniformly random element from S. If A is a randomized PPT algorithm, we write $y \xleftarrow{r} A(x;r)$ to denote that y is assigned the output of A on input x and random coins r. If we drop r, we assume that A is run on freshly drawn coins. When we write $(y;r) \xleftarrow{r} A(x)$, we mean that r is assigned the value of the fresh coins used by A. The notation $y \leftarrow A(x)$ is used for deterministic algorithms. A function $\epsilon : \mathbb{N} \to \mathbb{R}$ is negligible if $\epsilon(\tau) = \tau^{-\omega(1)}$. We denote the bit-wise exclusive OR (XOR) as $\oplus$. If m is a string, then $|m|$ denotes the length in bits of m. If S is a set, $|S|$ denotes the cardinality of S. If an argument is a list, we require that the list has an injective encoding, which allows to uniquely decode each element again. We also assume that the size of the encoding of the list as a bit string only depends on the length of the input, not the concrete values.

b) Trapdoor One-Way Permutations (TDP): Let $(f, f^{-1}, \Sigma) \xleftarrow{r} \mathsf{Gen}(1^\tau)$[1] be the instance generator for a deterministic function $f : \Sigma \to \Sigma$ defining a permutation over Σ, with a deterministic inversion function $f^{-1} : \Sigma \to \Sigma$, such that we have 1) for all $\tau \in \mathbb{N}$, for all $(f, f^{-1}, \Sigma) \xleftarrow{r} \mathsf{Gen}(1^\tau)$, and for all $x \in \Sigma$ we have $x = f^{-1}(f(x))$, and 2) for all PPT adversaries $\mathcal{A}$ we have $\Pr[(f, f^{-1}, \Sigma) \xleftarrow{r} \mathsf{Gen}(1^\tau), x \xleftarrow{r} \Sigma : x = \mathcal{A}(f, f(x), \Sigma)] \le \epsilon(\tau)$ for some negligible function ϵ. 3) We also require an algorithm Sample_Σ, which returns a uniformly sampled element from Σ. For simplicity, we define that $|\Sigma| \ge 2^{2\tau}$. An RSA-key generator $\mathsf{RSA.KGen}$ is a trapdoor one-way permutation under the RSA assumption with $\Sigma = \mathbb{Z}_N^*$, $f(x) = x^e \bmod N$ and $f^{-1}(y) = y^d \bmod N$ [5]. We use the convention that the randomness space of Sample_Σ has the same size as Σ.

c) Universal Composability: We assume that the reader is familiar with the universal composability (UC) framework [13], where security of a protocol is defined as an environment's inability to distinguish whether it is interacting with the real protocol through a real-world adversary, or with an ideal functionality through a simulator. Note, an ideal functionality returns control to the environment if it receives input on an interface not explicitly defined. If a protocol "ignores" an incoming network message, the party gives control back to the environment. The same is true for ideal functionalities, i.e., ignored inputs lead to an implicit output to the environment such that it receives back control. Note, if an ideal functionality gives some output, it directly gives up control. When we write "wait for input from $\mathcal{A}$", we require that the adversary immediately answers the request, as defined by Camenisch et al. [9]. Moreover, to avoid unhelpful boilerplate notation, we define that the adversary can only query these interfaces once for each request, and only if requested to do so. Finally, we require that a secret key always contains the corresponding public key.

[1]The instance generator only returns a description of Σ.

```
**Experiment** $\mathsf{Exp}^{\mathsf{IND\text{-}CCA2}}_{\mathsf{ENC},\mathcal{A}}(\tau)$:
$(epk, esk) \xleftarrow{r} \mathsf{KGen}(1^\tau)$
$b \xleftarrow{r} \{0,1\}$
$((m_0, m_1), \ell, state_{\mathcal{A}}) \xleftarrow{r} \mathcal{A}^{\mathrm{RO}(\cdot),\textsc{Dec}(\cdot,\cdot)}(epk)$
  where $\textsc{Dec}(\cdot)$ on input $(c, \ell)$:
    return $m' \leftarrow \mathsf{Dec}(esk, c, \ell)$
  where $\mathrm{RO}(\cdot)$ on input $s$:
    return $\mathcal{H}(s)$
If $|m_0| \neq |m_1| \vee m_0 \notin \mathcal{M} \vee m_1 \notin \mathcal{M}$:
  $c \leftarrow \perp$
Else:
  $c \xleftarrow{r} \mathsf{Enc}(epk, m_b, \ell)$
$b^* \xleftarrow{r} \mathcal{A}^{\mathrm{RO}(\cdot),\textsc{Dec}'(\cdot,\cdot)}(state_{\mathcal{A}}, c)$
  where $\textsc{Dec}'(\cdot,\cdot)$ behaves as $\textsc{Dec}(\cdot,\cdot)$,
  but returns $\perp$ if $(c, \ell)$ is queried.
return 1, if $b^* = b$
return 0
```

Fig. 1. Labeled IND-CCA2 for messages with arbitrary length.

III. Additional Security Definitions

A. Standard Definitions

a) IND-CCA2-Security (With Labels): The following definition is taken from [11], which itself in based on [12], [43]. A label can be seen as a public piece of information, which is non-malleably attached to a ciphertext. Definitions with and without labels are essentially equivalent.

Definition 1 (IND-CCA2-Security (With Labels)): An encryption scheme $\mathsf{ENC} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ for arbitrary length messages with labels is IND-CCA2-secure, if for all PPT adversaries $\mathcal{A}$, $|\Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CCA2}}_{\mathsf{ENC},\mathcal{A}}(\tau) = 1] - 1/2| \le \epsilon(\tau)$ for some negligible function ϵ, and the experiment given in Figure 1.

This definition requires that an adversary $\mathcal{A}$ cannot decide what plaintext a given ciphertext contains, even if $\mathcal{A}$ is allowed to query for decryptions with arbitrary labels.

B. One-Sided Definitions

For the following definitions, we use the following conventions. A variable $\mathbf{v}$ in bold face is a vector. $\mathcal{I}$ is an index set. If we write $\mathbf{v}_{\mathcal{I}}$ we mean that we address the elements in $\mathbf{v}$ indexed by $\mathcal{I}$.

a) RECV-SIM-Security: The following definition is plainly taken from [11].

Definition 2 (RECV-SIM-Security): An encryption scheme $\mathsf{NCE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is RECV-SIM-secure if for all PPT adversaries $\mathcal{A}$ there exists a stateful PPT simulator $\mathsf{SIM}_{\mathsf{NCE}}$ such that $\big|\Pr[\mathsf{Exp}^{\mathsf{RECV\text{-}SIM-real}}_{\mathsf{NCE},\mathcal{A}}(\tau) = 1] - \Pr[\mathsf{Exp}^{\mathsf{RECV\text{-}SIM-ideal}}_{\mathsf{NCE},\mathcal{A},\mathsf{SIM}_{\mathsf{NCE}}}(\tau) = 1]\big| \le \epsilon(\tau)$ for some negligible function ϵ and the experiments of Figure 2.
In a nutshell, this definition requires that an adversary cannot decide whether is sees simulated ciphertexts or real ones, even if it learns the secret decryption key at some point. In this definition, the adversary does never receive any randomness used.

```
Experiment $\mathsf{Exp}^{\mathsf{RECV\text{-}SIM-ideal}}_{\mathsf{NCE},\mathcal{A},\mathsf{SIM}_{\mathsf{NCE}}}(\tau)$:
  $epk \xleftarrow{r} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{publickey}, 1^\tau)$
  $\mathcal{Q} \leftarrow \emptyset$
  $state_{\mathcal{A}} \xleftarrow{r} \mathcal{A}^{\mathrm{RO}(\cdot),\textsc{Enc}(\cdot,\cdot),\textsc{Dec}(\cdot,\cdot)}(epk)$
    where $\textsc{Enc}(\cdot,\cdot)$ on input $(m,\ell)$:
      $c \xleftarrow{r} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{encrypt}, |m|, \ell)$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(c,m,\ell)\}$
      return $c$
    where $\textsc{Dec}(\cdot,\cdot)$ on input $(c,\ell)$:
      if $(c,m,\ell) \in \mathcal{Q}$, return $m$
      else, return $m \leftarrow \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{decrypt}, C, \ell)$
    where $\mathrm{RO}(\cdot)$ on input $s$:
      return $h_k \xleftarrow{r} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{roquery}, s)$
  $esk \xleftarrow{r} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{keyleak}, \mathcal{Q})$
  return $\mathcal{A}^{\mathrm{RO}(\cdot)}(esk, state_{\mathcal{A}})$
```

```
Experiment $\mathsf{Exp}^{\mathsf{RECV\text{-}SIM-real}}_{\mathsf{NCE},\mathcal{A}}(\tau)$:
  $(epk, esk) \xleftarrow{r} \mathsf{KGen}(1^\tau)$
  $state_{\mathcal{A}} \xleftarrow{r} \mathcal{A}^{\mathrm{RO}(\cdot),\textsc{Enc}(\cdot,\cdot),\textsc{Dec}(\cdot,\cdot)}(epk)$
    where $\textsc{Enc}(\cdot,\cdot)$ on input $(m,\ell)$:
      return $\mathsf{Enc}(epk, m, \ell)$
    where $\textsc{Dec}(\cdot,\cdot)$ on input $(c,\ell)$:
      return $\mathsf{Dec}(esk, c, \ell)$
    where $\mathrm{RO}(\cdot)$ on input $s$:
      return $\mathcal{H}(s)$
  return $\mathcal{A}^{\mathrm{RO}(\cdot)}(esk, state_{\mathcal{A}})$
```

Fig. 2. Experiments RECV-SIM-ideal and RECV-SIM-real for the RECV-SIM definition.

```
Experiment $\mathsf{Exp}^{\mathsf{SSIM\text{-}SO-ideal}}_{\mathsf{NCE},S,n}(\tau)$:
  $\mathsf{Dist} \xleftarrow{r} S(1^\tau)$
  $\mathbf{m} = (m_1, m_2, \ldots, m_n) \xleftarrow{r} \mathsf{Dist}_{1,2,\ldots,n}$
  $\mathcal{I} \xleftarrow{r} S()$
  $\mathsf{output} \xleftarrow{r} S(\mathbf{m}_{\mathcal{I}})$
  return $(\mathbf{m}, \mathsf{Dist}, \mathcal{I}, \mathsf{output})$

Experiment $\mathsf{Exp}^{\mathsf{SSIM\text{-}SO-real}}_{\mathsf{NCE},\mathcal{A},n}(\tau)$:
  $(epk, esk) \xleftarrow{r} \mathsf{KGen}(1^\tau)$
  $(\mathsf{Dist}, \mathsf{state}_1) \xleftarrow{r} \mathcal{A}(epk)$
  $\mathbf{m} = (m_1, m_2, \ldots, m_n) \xleftarrow{r} \mathsf{Dist}_{1,2,\ldots,n}$
  $(\mathbf{c}; \mathbf{r}) = ((c_1; r_1), (c_2; r_2), \ldots, (c_n; r_n)) \xleftarrow{r} \mathsf{Enc}(epk, m_i)_{1,2,\ldots,n}$
  $(\mathcal{I}, \mathsf{state}_2) \xleftarrow{r} \mathcal{A}(\mathbf{c}, \mathsf{state}_1)$
  $\mathsf{output} \xleftarrow{r} \mathcal{A}(\mathbf{r}_{\mathcal{I}}, \mathbf{m}_{\mathcal{I}}, \mathsf{state}_2)$
  return $(\mathbf{m}, \mathsf{Dist}, \mathcal{I}, \mathsf{output})$
```

Fig. 3. Experiments SSIM-SO-ideal and SSIM-SO-real for the SSIM-SO definition.

```
Experiment $\mathsf{Exp}^{\mathsf{RSIM\text{-}SO-ideal}}_{\mathsf{NCE},S,n}(\tau)$:
  $\mathsf{Dist} \xleftarrow{r} S(1^\tau)$
  $\mathbf{m} = (m_1, m_2, \ldots, m_n) \xleftarrow{r} \mathsf{Dist}_{1,2,\ldots,n}$
  $\mathcal{I} \xleftarrow{r} S()$
  $\mathsf{output} \xleftarrow{r} S(\mathbf{m}_{\mathcal{I}})$
  return $(\mathbf{m}, \mathsf{Dist}, \mathcal{I}, \mathsf{output})$

Experiment $\mathsf{Exp}^{\mathsf{RSIM\text{-}SO-real}}_{\mathsf{NCE},\mathcal{A},n}(\tau)$:
  $(\mathbf{pk}, \mathbf{sk}, \mathbf{r}) = ((epk_1, esk_1; r_1) \ldots, (epk_n, esk_n; r_n)) \xleftarrow{r}$
      $\mathsf{KGen}(1^\tau)_{1,2,\ldots,n}$
  $(\mathsf{Dist}, \mathsf{state}_1) \xleftarrow{r} \mathcal{A}(\mathbf{pk})$
  $\mathbf{m} = (m_1, m_2, \ldots, m_n) \xleftarrow{r} \mathsf{Dist}_{1,2,\ldots,n}$
  $\mathbf{c} = (c_1, c_2, \ldots, c_n) \xleftarrow{r} \mathsf{Enc}(epk_i, m_i)_{1,2,\ldots,n}$
  $(\mathcal{I}, \mathsf{state}_2) \xleftarrow{r} \mathcal{A}(\mathbf{c}, \mathsf{state}_1)$
  $\mathsf{output} \xleftarrow{r} \mathcal{A}(\mathbf{r}_{\mathcal{I}}, \mathbf{m}_{\mathcal{I}}, \mathsf{state}_2)$
  return $(\mathbf{m}, \mathsf{Dist}, \mathcal{I}, \mathsf{output})$
```

Fig. 4. Experiments RSIM-SO-ideal and RSIM-SO-real for the RSIM-SO definition.

b) SSIM-SO-Security: The following definition is plainly taken from [30].

Definition 3 (SSIM-SO-Security): An encryption scheme $\mathsf{NCE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is SSIM-SO-secure if for all PPT adversaries $\mathcal{A}$ there exists a stateful PPT simulator S such that for every binary-output distinguisher D and any n polynomial in τ we have $\big|\Pr[D(\mathsf{Exp}^{\mathsf{SSIM\text{-}SO-real}}_{\mathsf{NCE},\mathcal{A},n}(\tau)) = 1] - \Pr[D(\mathsf{Exp}^{\mathsf{SSIM\text{-}SO-ideal}}_{\mathsf{NCE},S,n}(\tau)) = 1]\big| \leq \epsilon(\tau)$ for some negligible function ϵ and the experiments of Figure 3.
In a nutshell, this definition requires that an adversary cannot decide whether is sees simulated ciphertexts or real ones, even if it learns the randomness used to generate the ciphertexts at some point.

c) RSIM-SO-Security: The following definition is taken from [30], but adjusted for our notation. Note, here the secret key also contains the randomness used to create it.

Definition 4 (RSIM-SO-Security): An encryption scheme $\mathsf{NCE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is RSIM-SO-secure if for all PPT adversaries $\mathcal{A}$ there exists a stateful PPT simulator S such that for every binary-output distinguisher D and any n polynomial in τ we have $\big|\Pr[D(\mathsf{Exp}^{\mathsf{RSIM\text{-}SO-real}}_{\mathsf{NCE},\mathcal{A},n}(\tau)) = 1] - \Pr[D(\mathsf{Exp}^{\mathsf{RSIM\text{-}SO-ideal}}_{\mathsf{NCE},S,n}(\tau)) = 1]\big| \leq \epsilon(\tau)$ for some negligible function ϵ and the experiments of Figure 4.
In a nutshell, this definition requires that an adversary cannot decide whether it sees simulated ciphertexts or real ones, even if it learns the secret decryption key (with randomness) at some point. The adversary does never receive any randomness used for encryptions.

d) IND-NCER-Security: The following definition is taken from [16], [30], but we have added state to the adversary.

Let $\mathsf{ENC}^* = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Enc}^*, \mathsf{Dec}, \mathsf{Open})$. Algorithm KGen, Enc, and Dec are a standard encryption scheme (without labels). The fake encryption scheme Enc^* outputs a ciphertext c^* and a trapdoor t. Given the secret key esk, the public key epk, the fake-ciphertext c^*, the trapdoor t, and a plaintext m, algorithm Open outputs esk^*. Refer to [30] for a formal definition, including correctness.

Definition 5 (IND-NCER-Security): An encryption scheme $\mathsf{ENC}^* = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Enc}^*, \mathsf{Dec}, \mathsf{Open})$ is IND-NCER-secure, if for all PPT adversaries $\mathcal{A}$, we have $|\Pr[\mathsf{Exp}^{\mathsf{IND\text{-}NCER}}_{\mathsf{ENC},\mathcal{A}}(\tau) = 1] - 1/2| \leq \epsilon(\tau)$ for some negligible function ϵ and the experiment given in Figure 5.

This definition requires that an adversary $\mathcal{A}$ cannot decide

```
**Experiment** $\mathsf{Exp}^{\mathsf{IND\text{-}NCER}}_{\mathsf{ENC},\mathcal{A}}(\tau)$:
$(epk, esk_0; r_{\mathsf{key}_0}) \xleftarrow{r} \mathsf{KGen}(1^\tau)$
$b \xleftarrow{r} \{0,1\}$
$(m, \mathsf{state}) \xleftarrow{r} \mathcal{A}(epk)$
$c_0 \xleftarrow{r} \mathsf{Enc}(epk, m)$
$(c_1, t) \xleftarrow{r} \mathsf{Enc}^*(epk, 1^{|m|})$
$(esk_1, r_{\mathsf{key}_1}) \xleftarrow{r} \mathsf{Open}(esk_0, epk, c_1, t, m)$
$b^* \xleftarrow{r} \mathcal{A}(\mathsf{state}, (esk_b, r_{\mathsf{key}_b}), c_b)$
return 1, if $b^* = b$
return 0
```

Fig. 5. IND-NCER-Security

whether is sees simulated secret key randomness (and ciphertext), or the real one, even when it receives an encryption of a message of its own choice.

IV. Property-Based Non-Committing Encryption

Most existing property-based security notions aim at standard-model instantiations and therefore have to circumvent Nielsen's impossibility result [45]. They either do so by encrypting only a single message under each public key (e.g., Canetti, Halevi, and Katz' IND-NCER notion [16]) or by considering only sender corruptions but no receiver corruptions (e.g., the SSIM-SO notion [30]), or only receiver corruptions but no sender corruptions (e.g., the RSIM-SO notion [30]) Camenisch et al.'s RECV-SIM notion [11] allows unlimited ciphertexts and receiver corruptions, but, because it focuses on a setting with secure erasures, does not give the adversary access to the randomness used in encryption or key generation.

We first introduce a new property-based security notion FULL-SIM for non-committing encryption. Unlike existing property-based notions [11], [30], the FULL-SIM adversary simultaneously has access to the randomness used in previous ciphertexts as well as to the randomness used to generate the key pair (and hence, the secret decryption key esk). We then provide a FULL-SIM secure NCE scheme based on trapdoor one-way permutations in the random-oracle model. We also study the relationship of FULL-SIM to existing notions, finding FULL-SIM to be either strictly stronger or incomparable.

A. Property-Based Definition of Non-Committing Encryption

A labeled non-committing encryption scheme $\mathsf{NCE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ consists of three algorithms. The first algorithm is the key generation algorithm $(epk, esk) \xleftarrow{r} \mathsf{KGen}(1^\tau)$, which outputs a public and the corresponding secret key. The public key implicitly specifies a message space $\mathcal{M}$. The encryption algorithm $c \xleftarrow{r} \mathsf{Enc}(epk, m, \ell)$ computes a ciphertext c on input of a public key epk, a message $m \in \mathcal{M}$, and a label $\ell \in \{0,1\}^*$. The deterministic decryption algorithm $m' \leftarrow \mathsf{Dec}(esk, c, \ell)$ takes as input a secret key esk, a ciphertext c and a label ℓ and outputs either a message m', or $\perp$ if decryption failed. Clearly, for definitions and schemes without labels, one can simply fix all labels to the empty string below.

The scheme must be correct, meaning that for all $\tau \in \mathbb{N}$, all $(epk, esk) \xleftarrow{r} \mathsf{KGen}(1^\tau)$, all messages $m \in \mathcal{M}$, and all labels $\ell \in \{0,1\}^*$, we have $m = \mathsf{Dec}(esk, \mathsf{Enc}(epk, m, \ell), \ell)$ with probability one.

We now define our notion of FULL-SIM-security for labeled non-committing encryption schemes NCE. While not made explicit in the notation, we allow $\mathsf{SIM}_{\mathsf{NCE}}$ to keep state between invocations. Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\tau$ be a random oracle [5].

Definition 6 (FULL-SIM-Security): An encryption scheme $\mathsf{NCE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is FULL-SIM-secure if for all PPT adversaries $\mathcal{A}$ with binary output there exists a stateful PPT simulator $\mathsf{SIM}_{\mathsf{NCE}}$ such that: $\big|\Pr[\mathsf{Exp}^{\mathsf{FULL\text{-}SIM\text{-}real}}_{\mathsf{NCE},\mathcal{A}}(\tau) = 1] - \Pr[\mathsf{Exp}^{\mathsf{FULL\text{-}SIM\text{-}ideal}}_{\mathsf{NCE},\mathcal{A},\mathsf{SIM}_{\mathsf{NCE}},\mathcal{L}}(\tau) = 1]\big| \le \epsilon(\tau)$ for some negligible function ϵ, and the experiments depicted in Figure 6.

Our definition says that an encryption scheme NCE is FULL-SIM-secure, if no PPT adversary $\mathcal{A}$ can distinguish between simulated ciphertexts and real ones. The adversary $\mathcal{A}$ receives full adaptive access to oracles for new encryptions, decryptions, the randomness used for encryptions, as well as the randomness used for generating the secret key. The simulated ciphertexts do not contain any information about the plaintext other than what is explicitly given to the simulator by the leakage function $\mathcal{L}$. In our case, we define that the leakage function $\mathcal{L} : \{0,1\}^* \to \mathbb{N}_0$ returns the (bit-)length of the message m in question, which is a reasonable leakage definition for our use case. Only when the adversary asks for the randomness used to generate a ciphertext or the secret key, does the simulator obtain the corresponding messages, upon which it must provide a consistent view to $\mathcal{A}$.

B. Instantiation

We now give a concrete instantiation for an encryption scheme NCE that is FULL-SIM-secure. Our construction is identical to the RECV-SIM-secure encryption scheme by Camenisch et al. [11], which, in turn, borrows ideas from Bellare and Rogaway [5] and Nielsen [45]. We recall it here and prove it secure under our stronger FULL-SIM notion.

Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\tau$ be a hash function, modeled as a random oracle. Further, we require an encoding scheme $\mathsf{EC} = (\mathsf{Ec}_\tau, \mathsf{Dc}_\tau)$ which allows to map arbitrary length messages to a list of blocks with fixed length. More precisely, let $\mathsf{Ec}_\tau : \{0,1\}^* \to (\{0,1\}^\tau)^+$ be a deterministic injective encoding function and $\mathsf{Dc}_\tau : (\{0,1\}^\tau)^+ \to \{0,1\}^*$ be the corresponding deterministic decoding function that returns $\perp$ if no valid preimage exists. We require that both functions are computable in polynomial time and that the output length of Ec_τ only depends on the length of its input, while Dc_τ and Ec_τ need to be perfectly correct, i.e., for all $\tau \in \mathbb{N}$, and all messages $m \in \{0,1\}^*$ we have $m = \mathsf{Dc}_\tau(\mathsf{Ec}_\tau(m))$ with probability one.

a) Algorithms: Let us describe the construction first, which is compact enough to be implemented in real protocols, which avoids the major obstacle for real-life deployment. Hence, the complexity is hidden inside the simulator $\mathsf{SIM}_{\mathsf{NCE}}$, which we give after describing our construction.

$\mathsf{KGen}(1^\tau)$: Generate a random trapdoor one-way permutation, i.e., $(f, f^{-1}, \Sigma) \xleftarrow{r} \mathsf{TDP.Gen}(1^\tau)$. Set $\mathcal{M} = \{0,1\}^*$. Output the public key $epk = (f, \Sigma)$, and $esk = f^{-1}$ as the secret key.

```
Experiment $\mathsf{Exp}^{\mathsf{FULL\text{-}SIM\text{-}ideal}}_{\mathsf{NCE},\mathcal{A},\mathsf{SIM}_{\mathsf{NCE}},\mathcal{L}}(\tau)$:
  $epk \xleftarrow{r} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{publickey}, 1^\tau)$
  $\mathcal{Q} \leftarrow \emptyset$
  return $\mathcal{A}^{\mathrm{RO}(\cdot),\textsc{Enc}(\cdot,\cdot),\textsc{Dec}(\cdot,\cdot),\textsc{Rand}(\cdot,\cdot),\textsc{GetSK}()}(epk)$
    where $\textsc{Enc}(\cdot,\cdot)$ on input $(m,\ell)$:
      if $\textsc{GetSK}()$ has been called:
        let $c \xleftarrow{r} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{encryptm}, m, \ell)$
      else:
        let $c \xleftarrow{r} \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{encryptl}, \mathcal{L}(m), \ell)$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(c,m,\ell)\}$
      return $c$
    where $\textsc{Dec}(\cdot,\cdot)$ on input $(c,\ell)$:
      if $(c,m,\ell) \in \mathcal{Q}$, return $m$
      else, return $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{decrypt}, c, \ell)$
    where $\mathrm{RO}(\cdot)$ on input $s$:
      return $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{roquery}, s)$
    where $\textsc{GetSK}()$:
      return $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{keyleak}, \mathcal{Q})$
    where $\textsc{Rand}(\cdot,\cdot)$ on input $(c,\ell)$:
      if $(c,m,\ell) \notin \mathcal{Q}$ for some $m$, return $\perp$
      else, return $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{randomness}, (c,m,\ell))$
```

```
Experiment $\mathsf{Exp}^{\mathsf{FULL\text{-}SIM\text{-}real}}_{\mathsf{NCE},\mathcal{A}}(\tau)$:
  $(epk, esk; r_{\mathsf{key}}) \xleftarrow{r} \mathsf{KGen}(1^\tau)$
  $\mathcal{Q} \leftarrow \emptyset$
  return $\mathcal{A}^{\mathrm{RO}(\cdot),\textsc{Enc}(\cdot,\cdot),\textsc{Dec}(\cdot,\cdot),\textsc{Rand}(\cdot,\cdot),\textsc{GetSK}()}(epk)$
    where $\textsc{Enc}(\cdot,\cdot)$ on input $(m,\ell)$:
      let $(c;r) \xleftarrow{r} \mathsf{Enc}(epk, m, \ell)$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(c,m,\ell,r)\}$
      return $c$
    where $\textsc{Dec}(\cdot,\cdot)$ on input $(c,\ell)$:
      return $\mathsf{Dec}(esk, c, \ell)$
    where $\mathrm{RO}(\cdot)$ on input $s$:
      return $\mathcal{H}(s)$
    where $\textsc{GetSK}()$:
      return $r_{\mathsf{key}}$
    where $\textsc{Rand}(\cdot,\cdot)$ on input $(c,\ell)$:
      if $(c,m,\ell,r) \notin \mathcal{Q}$, for some $r$ and $m$,
        return $\perp$
      else, return $r$
```

Fig. 6. Experiments FULL-SIM-ideal and FULL-SIM-real.

$\mathsf{Enc}(epk, m, \ell)$: Let $(m_1, m_2, \ldots, m_k) \leftarrow \mathsf{EC.Ec}_\tau(m)$. Draw $x \xleftarrow{r} \mathsf{TDP.Sample}_\Sigma(1^\tau)$, and compute $c_{(1)} \leftarrow \mathsf{TDP}.f(x)$, $c_{(i,2)} \leftarrow \mathcal{H}(i,x) \oplus m_i$ for all $0 < i \le k$, and $c_{(3)} \leftarrow \mathcal{H}(x,k,m,\ell)$. Output the ciphertext $c = (c_{(1)}, (c_{(1,2)}, c_{(2,2)}, \ldots, c_{(k,2)}), c_{(3)})$.

$\mathsf{Dec}(esk, c, \ell)$: Parse c as $(c_{(1)}, (c_{(1,2)}, c_{(2,2)}, \ldots, c_{(k',2)}), c_{(3)})$ for some $k' \ge 1$ with each $|c_{(i,2)}| = \tau$. Compute $x' \leftarrow \mathsf{TDP}.f^{-1}(c_{(1)})$ and $m'_i \leftarrow \mathcal{H}(i,x') \oplus c_{(i,2)}$ for all $0 < i \le k'$. Set $m' \leftarrow \mathsf{EC.Dc}_\tau(m'_1, m'_2, \ldots, m'_{k'})$. If $m' = \perp$ or $c_{(3)} \ne \mathcal{H}(x', k', m', \ell)$, output $\perp$, else output m'. This algorithm is deterministic.

The above construction clearly fulfills perfect correctness, if EC is perfectly correct.

C. Security of the Construction

We now prove that the above construction is FULL-SIM secure.

Theorem 1: The construction $\mathsf{NCE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ above is FULL-SIM-secure, if TDP.Gen is a secure trapdoor permutation generator and if $\mathcal{H}$ is modeled as a fully programmable and observable random oracle.

We first give a rough sketch to make the proof more understandable, and give the full proof afterwards.

Proof Sketch. Simplified, the proof works as follows. Decryption queries are answered by searching through the random-oracle table for entries $\mathcal{H}(i,x)$ and $\mathcal{H}(x,k,m,\ell)$ such that $c_{(1)} = \mathsf{TDP}.f(x)$, or returns $\perp$ if no such entries are found. When our simulator $\mathsf{SIM}_{\mathsf{NCE}}$ is asked to encrypt a message of a certain length, it returns a ciphertext with $c_{(1)} = \mathsf{TDP}.f(x)$ for a random x, but replacing $c_{(1,2)}, \ldots, c_{(k,2)}$ and $c_{(3)}$ with random strings. When the message m to which the ciphertext should decrypt becomes known, the simulator programs the random oracle $\mathcal{H}$ such that $\mathcal{H}(i,x) \oplus c_{(i,2)} = m_i$ and $c_{(3)} \leftarrow \mathcal{H}(x,k,m,\ell)$. This programming could fail if one of $\mathcal{H}(i,x)$ or $\mathcal{H}(x,k,m,\ell)$ was queried already. We show that this event BAD only happens with negligible probability.

Namely, there are three possibilities. The first is that $\mathsf{SIM}_{\mathsf{NCE}}$ already programmed one of these entries for a different ciphertext, meaning, the simulator itself accidentally drew the same x twice for different ciphertexts (denoted BADX). Due to the birthday bound, this only happens with negligible probability. The second case is that the programming of $\mathcal{H}(x,k,m,\ell)$ makes a ciphertext valid that the simulator previously had to decrypt and that it discarded as invalid. This essentially means that the adversary has to guess a random oracle output, which clearly happens only with negligible probability, denoted BADH. In the third case, the adversary made a random-oracle query involving x. Such an adversary can be turned into an inverter for the TDP.

Proof. We provide a simulator $\mathsf{SIM}_{\mathsf{NCE}}$ such that the view created by the simulator is indistinguishable from the view provided by the real experiment. The general idea is that the simulator honestly generates the key pair and honestly samples the first part $c_{(1)}$ of all ciphertexts c, but draws $c_{(2,i)}$ and $c_{(3)}$ randomly. Once the plaintext corresponding to a simulated ciphertext becomes known, i.e., when the adversary makes a $\textsc{Rand}$ or $\textsc{GetSK}$ query, the simulator programs the random oracle $\mathcal{H}$ such that the ciphertext actually decrypts to the correct message. As all the randomness used by the simulator is drawn honestly, this can be simulated as well, so we do not require secure erasures. The only thing that can make the simulation fail, is that the programming of the random oracle fails because entries have already been assigned. We show that such an event gives rise to an algorithm breaking the one-wayness of the TDP.

a) Description of the Simulator: The simulator $\mathsf{SIM}_{\mathsf{NCE}}$ keeps two initially empty lists $L_{\mathcal{H}}$, and $L_{\mathcal{R}}$. The first list $L_{\mathcal{H}}$ contains pairs (s, h), where h is the value returned by the random oracle on the query $\mathcal{H}(s)$. The second list $L_{\mathcal{R}}$ stores entries (c, m, ℓ, r, x) to keep track of the relationship between (simulated and non-simulated) ciphertexts c, messages m, labels ℓ, sampling randomness r and sampled values x.

Key Generation. On input of $(\mathsf{publickey}, 1^\tau)$, $\mathsf{SIM}_{\mathsf{NCE}}$ honestly generates the key pair, i.e., $((f, f^{-1}, \Sigma); r_{\mathsf{key}}) \xleftarrow{r} \mathsf{TDP.Gen}(1^\tau)$. It stores $esk = f^{-1}$, and r_{key}. $\mathsf{SIM}_{\mathsf{NCE}}$ returns (f, Σ) as epk.

Ramdom-Oracle Queries. For each query $(\mathsf{roquery}, s)$, $\mathsf{SIM}_{\mathsf{NCE}}$ checks whether there is an entry $(s, h) \in L_{\mathcal{H}}$ for some h. If so, $\mathsf{SIM}_{\mathsf{NCE}}$ returns h. Else, $\mathsf{SIM}_{\mathsf{NCE}}$ draws $h \xleftarrow{r} \{0,1\}^\tau$, adds (s, h) to $L_{\mathcal{H}}$, and returns h.

Encryption. Depending on the type of input, we need to branch:

- On input $(\mathsf{encryptm}, m, \ell)$, $\mathsf{SIM}_{\mathsf{NCE}}$ encrypts honestly, i.e., it runs $(c; r) \xleftarrow{r} \mathsf{NCE.Enc}(epk, m, \ell)$, using $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{roquery}, s)$ for random-oracle calls $\mathcal{H}(s)$, adds $(c, m, \ell, r, \perp_x)$ to $L_{\mathcal{R}}$, and returns c.
- On input $(\mathsf{encryptl}, \mathcal{L}(m) = |m|, \ell)$, $\mathsf{SIM}_{\mathsf{NCE}}$ draws $(x; r) \xleftarrow{r} \mathsf{TDP.Sample}_\Sigma(1^\tau)$. Let $k \leftarrow |\mathsf{EC.Ec}_\tau(1^{|m|})|/\tau$. It sets $c_{(1)} \leftarrow \mathsf{TDP}.f(x)$ and chooses $c_{(2,i)} \xleftarrow{r} \{0,1\}^\tau$ for $1 \leq i \leq k$ and $c_{(3)} \xleftarrow{r} \{0,1\}^\tau$. It adds $(c, \perp_m, \ell, r, x)$ to $L_{\mathcal{R}}$ and returns c.

Decryption. On input of $(\mathsf{decrypt}, c, \ell)$, $\mathsf{SIM}_{\mathsf{NCE}}$ computes $x' \leftarrow \mathsf{TDP}.f^{-1}(c_{(1)})$, lets $m'_i \leftarrow c_{(2,i)} \oplus \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{roquery}, (i, x'))$ for $i = 1, \ldots, k$, and computes $m' \leftarrow \mathsf{EC.Dc}_\tau(m'_1, m'_2, \ldots, m'_k)$. If $m' = \perp$ or $c_{(3)} \neq \mathsf{SIM}_{\mathsf{NCE}}(\mathsf{roquery}, (x', k, m', \ell))$, it returns $\perp$, otherwise it returns m'. Note that $\mathsf{SIM}_{\mathsf{NCE}}$ never receives a request where it has to decrypt a simulated ciphertext.

Encryption Randomness. On input of $(\mathsf{randomness}, (c, m, \ell))$, look up a tuple $(c, m', \ell, r, x) \in L_{\mathcal{R}}$ where $m' \in \{m, \perp_m\}$. Note that, from the description of experiment FULL-SIM-ideal, such a tuple always exists. If $m' = m$, then $\mathsf{SIM}_{\mathsf{NCE}}$ simply returns r. If $m' = \perp_m$, then $\mathsf{SIM}_{\mathsf{NCE}}$ must first program $\mathcal{H}$ to ensure consistent encryption, as with the given randomness r the adversary can re-encrypt m to see whether it obtains c. It does so as follows. Let $(m_1, m_2, \ldots, m_k) \leftarrow \mathsf{EC.Ec}_\tau(m)$. If $((x, k, m, \ell), c'_{(3)}) \in L_{\mathcal{H}}$ for some $c'_{(3)}$ or if $((i, x), h_i) \in L_{\mathcal{H}}$ for some $1 \leq i \leq k$, then we say that event BAD happened and $\mathsf{SIM}_{\mathsf{NCE}}$ aborts. Otherwise, $\mathsf{SIM}_{\mathsf{NCE}}$ adds $((i, x), m_i \oplus c_{(2,i)})$ for $1 \leq i \leq k$ as well as $((x, k, m, \ell), c_{(3)})$ to $L_{\mathcal{H}}$. It then updates $(c, \perp_m, \ell, r, x)$ to (c, m, ℓ, r, x) in $L_{\mathcal{R}}$ and returns r.

Key Leakage. On input $(\mathsf{keyleak}, \mathcal{Q})$, $\mathsf{SIM}_{\mathsf{NCE}}$ first programs the random oracle $\mathcal{H}$ to ensure consistent decryption for all $(c, m, \ell) \in Q$ in the same way as for answering RAND queries above. $\mathsf{SIM}_{\mathsf{NCE}}$ then returns the random coins r_{key} used to generate the secret key esk.

From the way random oracles are programmed, it is clear that the simulation is perfect unless the event BAD happens. Using Pr[REAL] and Pr[IDEAL] as shorthand notations for the probability that the experiment outputs 1 for the real and ideal experiments, respectively, we have that

$$\begin{aligned}
&\big|\Pr[\mathsf{REAL}] - \Pr[\mathsf{IDEAL}]\big| \\
&\quad = \big|(\Pr[\mathsf{REAL}|\mathsf{BAD}] - \Pr[\mathsf{IDEAL}|\mathsf{BAD}]) \cdot \Pr[\mathsf{BAD}] \\
&\qquad + (\Pr[\mathsf{REAL}|\overline{\mathsf{BAD}}] - \Pr[\mathsf{IDEAL}|\overline{\mathsf{BAD}}]) \cdot \Pr[\overline{\mathsf{BAD}}]\big| \\
&\quad = \big|(\Pr[\mathsf{REAL}|\mathsf{BAD}] - \Pr[\mathsf{IDEAL}|\mathsf{BAD}])\big| \cdot \Pr[\mathsf{BAD}] \quad (1)\\
&\quad \leq \Pr[\mathsf{BAD}] \ , \quad (2)
\end{aligned}$$

where (1) is true because $\Pr[\mathsf{REAL} \mid \overline{\mathsf{BAD}}] - \Pr[\mathsf{IDEAL} \mid \overline{\mathsf{BAD}}] = 0$ and (2) is true because the first factor of (1) is at most one.

b) Reduction from Trapdoor Permutations: We have left to prove that event BAD happens with negligible probability. We do so by proving that any adversary that causes BAD to occur gives rise to an algorithm breaking the one-wayness of the trapdoor permutation. The reduction is similar to the analysis of Camenisch et al. [11], but is tighter and explicitly gives out the randomness used for encryption to the adversary.

Let q_e be the number of encryption queries, q_d the number of decryption queries, and q_h the number of random-oracle queries to $\mathcal{H}$. Assume towards contradiction that $\Pr[\mathsf{BAD}] > \epsilon(\tau)$. We can then construct an algorithm $\mathcal{B}$ which outputs the preimage of a TDP challenge point y with non-negligible probability. Algorithm $\mathcal{B}$ receives (f, y, Σ) as input from the TDP challenger. It then interacts with $\mathcal{A}$ as follows.

Key Generation. On input of $(\mathsf{publickey}, 1^\tau)$, $\mathcal{B}$ draws $j \xleftarrow{r} \{1, 2, \ldots, q_e\}$, and returns (f, Σ) as epk.

RO Queries. On input $(\mathsf{roquery}, s)$, $\mathcal{B}$ checks if there is an entry $(s, h) \in L_{\mathcal{H}}$ for some h. If so, $\mathcal{B}$ returns h. Else, $\mathcal{B}$ draws $h \xleftarrow{r} \{0,1\}^\tau$.

If s is of the form (x, k, m, ℓ) and there exists a previously rejected ciphertext that, by assigning h as the output of $\mathcal{H}(s)$, should have been considered valid, then we say that event BADH happened and $\mathcal{B}$ aborts. More precisely, if there exists a tuple $(c, \ell) \in \mathcal{L}_c$ with $c = (c_{(1)}, c_{(1,2)}, \ldots, c_{(k,2)}, c_{(3)})$ such that $c_{(1)} = \mathsf{TDP}.f(x)$, $c_{(3)} = h$, and $m = \mathsf{EC.Dc}_\tau(c_{(1,2)} \oplus \mathcal{H}(1, x), \ldots, c_{(k,2)} \oplus \mathcal{H}(k, x))$, then BADH happened and $\mathcal{B}$ aborts, whereby random-oracle queries $\mathcal{H}(i, x)$ are simulated as described here.

Otherwise, $\mathcal{B}$ adds (s, h) to $L_{\mathcal{H}}$ and returns h.

Encryption. On input $(\mathsf{encryptl}, |m|, \ell)$, it proceeds as follows. If this is the jth encryption query, then $\mathcal{B}$ sets $c_{(1)} \leftarrow y$ and $x, r \leftarrow \perp$. Otherwise, it draws $(x; r) \xleftarrow{r} \mathsf{TDP.Sample}_\Sigma(1^\tau)$, sets $c_{(1)} \leftarrow \mathsf{TDP}.f(x)$, and tests whether $x \in \mathcal{L}_x$. If so, then we say that event BADX happened and $\mathcal{B}$ aborts, otherwise $\mathcal{B}$ adds x to $\mathcal{L}_x$. Let $k \leftarrow |\mathsf{EC.Ec}_\tau(1^{|m|})|/\tau$. Algorithm $\mathcal{B}$ chooses $c_{(2,i)} \xleftarrow{r} \{0,1\}^\tau$ for all $1 \leq i \leq k$ and $c_{(3)} \xleftarrow{r} \{0,1\}^\tau$, adds $(c, \perp_m, \ell, r, x)$ to $L_{\mathcal{R}}$, and returns c. Note, $(\mathsf{encryptm}, m, \ell)$ is never received.

Decryption. On input $(\mathsf{decrypt}, c, \ell)$ from $\mathcal{A}$, $\mathcal{B}$ proceeds as follows. Look for an entry $((x, k, m, \ell), c_{(3)}) \in \mathcal{L}_{\mathcal{H}}$ such that $\mathsf{TDP}.f(x) = c_{(1)}$ and $m = \mathsf{EC.Dc}_\tau(m_1, m_2, \ldots, m_k)$ for $m_i \leftarrow c_{(2,i)} \oplus \mathcal{H}(i, x)$, where calls to $\mathcal{H}$ are simulated as above. Note that at most one such entry can exist because, TDP being a permutation, there exists only one $x \in \Sigma$ such that $\mathsf{TDP}.f(x) = c_{(1)}$, which then unique defines $m_1, \ldots, m_k$ and thereby m. If no such entry exists, then $\mathcal{B}$ adds (c, ℓ) to $\mathcal{L}_c$ and returns $\perp$, otherwise it returns m.

Encryption Randomness. On input of $(\mathsf{randomness}, (c, m, \ell))$, look up a tuple $(c, m', \ell, r, x) \in L_{\mathcal{R}}$ for $m' \in \{m, \perp_m\}$. If $m' = m$ then $\mathcal{B}$ returns r. If $m' = \perp_m$ then $\mathcal{B}$ must program $\mathcal{H}$ to ensure consistent encryption. If now the conditions of the BAD event in $\mathsf{SIM}_{\mathsf{NCE}}$ are satisfied, i.e., if there exists a tuple $((x', k, m, \ell), \cdot) \in L_{\mathcal{H}}$ or a tuple $((i, x'), \cdot) \in L_{\mathcal{H}}$ such that $\mathsf{TDP}.f(x') = c_{(1)}$, and if additionally c is the j-th simulated ciphertext, i.e., if $x = r = \perp$, then $\mathcal{B}$ outputs x' as its preimage for $y = c_{(1)}$. If the conditions for BAD are satisfied but c is not the j-th ciphertext, then $\mathcal{B}$ aborts. If the conditions for BAD are not satisfied, then $\mathcal{B}$ programs the random oracle in the same way as $\mathsf{SIM}_{\mathsf{NCE}}$, i.e., by adding $((i, x), m_i \oplus c_{(2,i)})$ for $1 \leq i \leq k$ as well as $((x, k, m, \ell), c_{(3)})$ to $L_{\mathcal{H}}$. Algorithm $\mathcal{B}$ updates $(c, \perp_m, \ell, r, x)$ to (c, m, ℓ, r, x) in $L_{\mathcal{R}}$ and returns r.

Key Leakage. On input of $(\mathsf{keyleak}, \mathcal{Q})$, $\mathcal{B}$ checks each entry (i, x) and (x, k, m, ℓ) in $\mathcal{L}_{\mathcal{H}}$ whether $f(x) = y$. If so, then $\mathcal{B}$ returns x as the preimage of y to the challenger. Otherwise, it aborts.

Algorithm $\mathcal{B}$ succeeds in inverting y with probability $1/q_e$ whenever event BAD happens and neither BADH nor BADX happen, i.e.

$$\begin{aligned} \Pr[\mathcal{B} \text{ succeeds}] &= \frac{1}{q_e} \Pr[\mathsf{BAD} \wedge \overline{\mathsf{BADH}} \wedge \overline{\mathsf{BADX}}] \\ &\geq \frac{1}{q_e} \big(\Pr[\mathsf{BAD}] - \Pr[\mathsf{BADH}] - \Pr[\mathsf{BADX}] \big) \end{aligned}$$

Note that not being able to respond to a $\mathsf{keyleak}$ input does not influence the success probability of $\mathcal{B}$, because after a $\mathsf{keyleak}$ input the BAD event can no longer happen anyway.

The event BADH happens when the response to a new random-oracle query $\mathcal{H}(x, k, m, \ell)$ turns a ciphertext that was previously rejected during a decryption query into a valid ciphertext. For each decryption query $\textsc{Dec}(c, \ell)$, there is only a single random-oracle query $\mathcal{H}(x, k, m, \ell')$ that could cause BADH to happen though, namely the query where $\mathsf{TDP}.f(x) = c_{(1)}$, k is the number of blocks in $c_{(2)} = (c_{(1,2)}, \ldots, c_{(k,2)})$, $m = \mathsf{EC.Dc}_\tau(c_{(1,2)} \oplus \mathcal{H}(1, x), \ldots, c_{(k,2)} \oplus \mathcal{H}(k, x))$, and $\ell' = \ell$. Each of those random-oracle queries $\mathcal{H}(x, k, m, \ell')$ has probability $1/2^\tau$ to hit $c_{(3)}$, so the overall probability of BADH is at most $\Pr[\mathsf{BADH}] \leq q_d/2^\tau$.

The event BADX happens when during encryption, a randomly chosen value $x \overset{\text{r}}{\leftarrow} \Sigma$ hits one of the at most q_e elements of $\mathcal{L}_{\mathcal{R}}$. Since $|\Sigma| \geq 2^{2\tau}$, the probability of this happening is $\Pr[\mathsf{BADX}] \leq q_e^2/2^{2\tau}$. Putting everything together, if $\epsilon'(t)$ is the maximum advantage of a PPT algorithm to break the one-wayness of TDP, then we have that $|\Pr[\mathsf{REAL}] - \Pr[\mathsf{IDEAL}]| \leq q_e \epsilon'(t) + \frac{q_d}{2^\tau} + \frac{q_e^2}{2^{2\tau}}$ which proves the theorem. $\square$

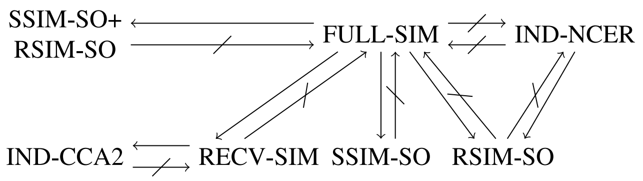

Fig. 7. Implications of security definitions. Solid arrows denote strict implications, while striked out arrows denote separations.

We stress that Nielsen's underlying construction [45] is not a FULL-SIM-secure encryption scheme, as it is trivially malleable. Refer to Appendix A for his construction.

The following corollary immediately follows from our construction.

Corollary 1: If trapdoor one-way permutations exist, then FULL-SIM-secure encryption schemes exist in the (fully programmable and observable) random-oracle model without erasures.

V. Relationships Between Security Notions

As depicted in Figure 7, we show that our definition is strictly stronger than the state-of-the-art definitions RECV-SIM [11], SSIM-SO [30], and RSIM-SO [30], while being incomparable to IND-NCER [16], [30].

Clearly, the RECV-SIM notion of Camenisch et al. [11], is the basis of our FULL-SIM notion, but does not give the adversary access to encryption or key generation randomness. As one would expect, our notion implies RECV-SIM. Since Camenisch et al. already proved that RECV-SIM security implies IND-CCA2 security [11], it automatically follows that FULL-SIM security also implies IND-CCA2 security.

The proofs of the following propositions are given in Appendix B.

Proposition 1 ($\mathsf{FULL\text{-}SIM} \implies \mathsf{RECV\text{-}SIM}$*):* Any encryption scheme that is FULL-SIM secure is also RECV-SIM secure.

Next, we focus on the chosen-plaintext definitions given by Hazay, Patra, and Warinschi [30]. As RSIM-SO and SSIM-SO do not incorporate labels, we define that the encryption and decryption oracles only accept empty labels, as the notions are essentially equivalent [49]. First, we address RSIM-SO security.

Proposition 2 ($\mathsf{FULL\text{-}SIM} \implies \mathsf{RSIM\text{-}SO}$*):* Any public-key encryption scheme that is FULL-SIM secure is also RSIM-SO secure.

Proposition 3 ($\mathsf{FULL\text{-}SIM} \implies \mathsf{SSIM\text{-}SO}$*):* Any public-key encryption scheme that is FULL-SIM secure is also SSIM-SO secure.

Next, we need some additional results to show the separations in the other direction, i.e., to show that our FULL-SIM definition is strictly stronger than RECV-SIM, RSIM-SO, and SSIM-SO.

Theorem 2: There is no FULL-SIM-secure NCE in the standard model.

Proof. This follows by construction. In the full version, we show how to realize the round-optimal secure message transfer functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{SMT}}$ using our functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$, which is black-box realized by any FULL-SIM-secure encryption scheme. The theorem follows by plugging in the impossibility result by Nielsen [45]. □

This means, that there is no standard model model instantiation of FULL-SIM-secure encryption schemes. Hence, the above implications are strict, as for the other notions there exist standard model instantiations under reasonable assumptions.

Note, this also means that we cannot avoid programmability of the random oracle in our construction, which also follows from the results given by Nielsen [45]. Thus, the global random-oracle (as defined by Canetti et al. [18]) cannot be used, and not even a CRS helps.

Proposition 4: If one-way trapdoor permutations exist, and the Decisional composite residuosity [46] (DCR) assumption holds, then IND-NCER security is incomparable to FULL-SIM security in the random oracle model. Meaning, there exists a scheme that is IND-NCER secure but not FULL-SIM secure and vice versa.

Moreover, even SSIM-SO-Security and RSIM-SO-Security together do not imply FULL-SIM-Security, as the adversary cannot proceed adaptively in the SSIM-SO game, i.e., the distribution is fixed, which is not the case in the FULL-SIM-Security game.

*Proposition 5 (*RECV-SIM $\not\Rightarrow$ FULL-SIM*):* If perfectly binding commitments exist, then there exists a scheme that is RECV-SIM secure but not FULL-SIM secure.

VI. Universally Composable Non-Committing Encryption

In this section, we introduce an ideal functionality in the UC framework for non-committing encryption $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$. We show that $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ and FULL-SIM security are essentially equivalent, in the sense that any FULL-SIM-secure scheme immediately gives rise to a secure instantiation of $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ and vice versa. We also show that $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ can be used to instantiate the secure message transmission functionality $\mathcal{F}_{\mathsf{SMT}}$, so that, as an immediate consequence, Nielsen's impossibility result [45] excludes any secure instantiations of $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ (and therefore, of FULL-SIM-secure encryption schemes) in the standard model.

A. Ideal Functionality for Non-Committing Encryption

The ideal functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ is depicted in Figure 8. In a nutshell, $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ encapsulates *local* public-key encryption, which is one of the most basic operations in modern cryptography. The functionality is based on the $\mathcal{F}_{\mathsf{AFSE}}$ functionality of Canetti et al. [16], but without the a-priori bound on the ciphertexts to be generated, while we also support labels. We therefore neither need any update interfaces, nor any special corruption interfaces. Compared to the definition by Canetti et al. [19], we also enforce that once a ciphertext is decrypted, the decryption remains fixed.

Encryption and decryption in our functionality are local operations that generate and decrypt actual ciphertexts. The ciphertexts are provided by the adversary $\mathcal{A}$ which, as long as the owner of the key pair is honest, only receives the information explicitly provided by the leakage function $\mathcal{L}$. We use the notation "send x to $\mathcal{A}$ and wait for y from $\mathcal{A}$" as a shorthand notation for requests to responsive environments as defined by Camenisch et al. [9], so that the functionality stalls until $\mathcal{A}$ provides a response y through a dedicated interface. While the functionality waits for a message from $\mathcal{A}$, the adversary cannot invoke any other interfaces of the ideal functionality, generate any network traffic, or activate or corrupt parties.

Previous functionalities in the literature require the adversary to provide an encryption algorithm that the functionality runs to generate ciphertexts for the encrypted messages [13], [17], [39], [40], [41]. All realizations using this paradigm suffer from the selective de-commitment problem [13], [40] or inherently require static adversaries. We avoid this problem by querying the adversary for each ciphertext that needs to be generated.

The public key for an instance of $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ is also provided by the adversary. To avoid implying certified keys, the encryption interface also accepts queries for different public keys than the registered key of this instance. Decryption for simulated ciphertexts works as expected, in the sense that for every ciphertext generated by the encryption interface, the functionality returns the original message without any involvement of the adversary. If, however, a ciphertext was not honestly generated, the functionality asks the adversary to provide the decryption.

a) The Interfaces: We briefly explain the interfaces of the $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ functionality.

- KeyGeneration. This interface can only be called once and allows the key-pair owner $\mathcal{P}$ to generate a key pair. The public key pk is provided by the adversary.
- Encryption. This interface allows any party, including the key-pair owner itself, to encrypt a message of arbitrary length. If the public key that is given as part of the input equals the public key stored for this instance, then the adversary only receives the information explicitly given by the leakage function $\mathcal{L}$, as long as $\mathcal{P}$ is honest. Otherwise, the functionality does not give any security guarantees, and gives the adversary the plaintext. Note, the adversary does not learn which party wants to encrypt, while the provided pk may also be adversarially chosen. The adversary must provide fresh ciphertexts for the stored public key and cannot overwrite ciphertexts.
- Decryption. This interface allows the key-pair owner to decrypt a given ciphertext. For simulated ciphertexts and ciphertexts that were decrypted before, the corresponding plaintexts are directly returned by the functionality. All other ciphertexts are sent to the adversary which needs to provide a decryption. Decryption is consistent, in the sense that once

1) `Key Generation`. On input (KEYGEN, sid) from party $\mathcal{P}$:
   - If $sid \neq (\mathcal{P}, sid')$ or a record $(\mathsf{keyrec}, \cdot)$ exists, ignore.
   - Send (KEYGEN, sid) to $\mathcal{A}$ and wait for $(\mathsf{KEYCONF}, sid, pk)$ from $\mathcal{A}$.
   - If $pk = \perp$, ignore.
   - Create record (keyrec, pk) and output $(\mathsf{KEYCONF}, sid, pk)$ to $\mathcal{P}$.
2) `Encryption`. On input $(\mathsf{ENCRYPT}, sid, pk, m, \ell)$ from party $\mathcal{Q}$:
   - If $pk = \perp$, ignore.
   - If no record (keyrec, pk) exists, send $(\mathsf{ENCRYPTM}, sid, pk, m, \ell)$ to $\mathcal{A}$ and wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
   - Else, if $sid = (\mathcal{P}, sid')$ and $\mathcal{P}$ is corrupt, send $(\mathsf{ENCRYPTM}, sid, pk, m, \ell)$ to $\mathcal{A}$ and wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
   - Else, send $(\mathsf{ENCRYPTL}, sid, pk, \mathcal{L}(m), \ell)$ to $\mathcal{A}$ and wait for $(\mathsf{CIPHERTEXT}, sid, c)$ from $\mathcal{A}$.
   - If there is a record (keyrec, v), let $pk' \leftarrow v$, else let $pk' \leftarrow \perp$.
   - If there is a record $(\mathsf{encryptionrec}, sid, pk, \cdot, \ell, c)$ and $pk = pk'$, ignore.
   - If there is a record $(\mathsf{decryptionrec}, sid, \cdot, \ell, c)$ and $pk = pk'$, ignore.
   - Create record $(\mathsf{encryptionrec}, sid, pk, m, \ell, c)$.
   - Output $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, pk)$ to $\mathcal{Q}$.
3) `Decryption`. On input $(\mathsf{DECRYPT}, sid, c, \ell)$ from party $\mathcal{P}$:
   - If $sid \neq (\mathcal{P}, sid')$, ignore.
   - If no record (keyrec, pk) exists, ignore.
   - If a record $(\mathsf{encryptionrec}, sid, pk, m, \ell, c)$ exists, output $(\mathsf{PLAINTEXT}, sid, c, m, \ell)$ to $\mathcal{P}$.
   - If a record $(\mathsf{decryptionrec}, sid, m, \ell, c)$ exists, output $(\mathsf{PLAINTEXT}, sid, c, m, \ell)$ to $\mathcal{P}$.
   - Send $(\mathsf{DECRYPT}, sid, c, \ell)$ to $\mathcal{A}$ and wait for $(\mathsf{PLAINTEXT}, sid, m)$ from $\mathcal{A}$.
   - Create record $(\mathsf{decryptionrec}, sid, m, \ell, c)$.
   - Output $(\mathsf{PLAINTEXT}, sid, c, m, \ell)$ to $\mathcal{P}$.

Fig. 8. Our functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ with leakage function $\mathcal{L}$.

a ciphertext is mapped to a given plaintext, the result of decrypting that ciphertext will always be the same.

B. *Instantiation of $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$*

Our construction is a wrapper around any FULL-SIM-secure labeled non-committing encryption scheme $\mathsf{NCE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ to match the input and output behavior of the ideal functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$, using the label to bind a ciphertext to a session sid. Any calls that NCE makes to a random oracle are relayed to a local instance of the random-oracle functionality $\mathcal{F}_{\mathsf{RO}}$ with session identifier $(sid, \mathcal{F}_{\mathsf{RO}})$. The definition of $\mathcal{F}_{\mathsf{RO}}$ can be found in the full version of this paper.

`Key Generation`. On input of (KEYGEN, sid), check that $sid = (\mathcal{P}, sid')$, and $\mathcal{P}$ is the identity. If this is not the case, ignore. If a record (key, sid, epk, esk) exists, ignore. Generate $(epk, esk) \overset{r}{\leftarrow} \mathsf{KGen}(1^\tau)$, store (key, sid, epk, esk), and output $(\mathsf{KEYCONF}, sid, epk)$.

`Encryption`. On input of $(\mathsf{ENCRYPT}, sid, pk, m, \ell)$, ignore if $pk = \perp$, or $sid \neq (\mathcal{P}, sid')$. Generate $c \overset{r}{\leftarrow} \mathsf{Enc}(pk, m, (\ell, sid))$. Output $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, pk)$.

`Decryption`. On input of $(\mathsf{DECRYPT}, sid, c, \ell)$, check that $sid = (\mathcal{P}, sid')$. If this is not the case, or no record (key, sid, epk, esk) exists, ignore. Otherwise, let $m \leftarrow \mathsf{Dec}(esk, c, (\ell, sid))$. Output $(\mathsf{PLAINTEXT}, sid, c, m, \ell)$.

Theorem 3: If NCE is FULL-SIM secure, then the above protocol securely realizes $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ in the $\mathcal{F}_{\mathsf{RO}}$-hybrid model without secure erasures and with adaptive corruptions.

Proof. By the FULL-SIM security of NCE, there must exist a simulator $\mathsf{SIM}_{\mathsf{NCE}}$ so that no PPT adversary exists that can distinguish between the real FULL-SIM game and the ideal FULL-SIM game with $\mathsf{SIM}_{\mathsf{NCE}}$. Given such a simulator $\mathsf{SIM}_{\mathsf{NCE}}$, we now describe a UC simulator SIM for the above instantiation of $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$. We subsequently show that any environment that can distinguish whether it is interacting with the real protocol and a real-world adversary $\mathcal{A}$ or with $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ and SIM can be used to build a FULL-SIM distinguisher for $\mathsf{SIM}_{\mathsf{NCE}}$, contradicting the FULL-SIM security of NCE.

a) Simulator: Given a FULL-SIM simulator $\mathsf{SIM}_{\mathsf{NCE}}$ and a real-world UC adversary $\mathcal{A}$, consider the following UC simulator SIM for $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$.

`Key Generation`. On input (KEYGEN, sid), the simulator SIM calls $\mathsf{SIM}_{\mathsf{NCE}}$ with $(\mathsf{publickey}, 1^\tau)$, which returns epk. SIM then records (key, sid, epk) and sends $(\mathsf{KEYCONF}, sid, epk)$ to $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$.

`Encryption`. On input $(\mathsf{ENCRYPTM}, sid, pk, m, \ell)$, the simulator SIM computes $(c; r) \overset{r}{\leftarrow} \mathsf{Enc}(pk, m, (\ell, sid))$, creates a record $(\mathsf{encryptionrec}, sid, c, m, \ell, r, pk, \mathsf{false})$, and sends $(\mathsf{CIPHERTEXT}, sid, c)$ to $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$. On input $(\mathsf{ENCRYPTL}, sid, pk, \mathcal{L}(m), \ell)$, SIM calls $\mathsf{SIM}_{\mathsf{NCE}}$ with $(\mathsf{encryptl}, \mathcal{L}(m), (\ell, sid))$ to $\mathsf{SIM}_{\mathsf{NCE}}$ to receive c. It then creates a record $(\mathsf{encryptionrec}, sid, c, \perp_m, \ell, \perp_r, pk, \mathsf{false})$ and sends $(\mathsf{CIPHERTEXT}, sid, c)$ to $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$. Note that this case implies $pk = epk$.

`Decryption`. On input $(\mathsf{DECRYPT}, sid, c, \ell)$, SIM calls $\mathsf{SIM}_{\mathsf{NCE}}$ with $(\mathsf{decrypt}, c, (\ell, sid))$ to obtain m and sends $(\mathsf{DECRYPT}, sid, m)$ to $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$.

`Random oracle`. In the $\mathcal{F}_{\mathsf{RO}}$-hybrid model, SIM must also provide responses to all parties' inputs to the $\mathcal{F}_{\mathsf{RO}}$ functionality. On input $(\mathsf{ROQUERY}, sid', s)$ from a party $\mathcal{P}$, SIM outputs $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{roquery}, s)$ if $sid' = (sid, \mathcal{F}_{\mathsf{RO}})$, or runs the actual code of $\mathcal{F}_{\mathsf{RO}}$ otherwise.

`Corruption`. We work with the standard corruption definition [13] where, upon corruption of a party, the simulator receives all previous inputs and outputs of that party and needs to provide a consistent view of the real-world state for that party to the real-world adversary $\mathcal{A}$. We need to consider different cases depending on which party gets corrupted.

If the holder of the secret key $\mathcal{P}$ becomes corrupted, SIM receives six lists corresponding to the inputs and outputs of the three interfaces of $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ to and from $\mathcal{P}$. The simulator SIM has to provide a realistic snapshot of $\mathcal{P}$'s real-world state to the adversary $\mathcal{A}$ that must contain, apart from the list of previous inputs and outputs that SIM just obtained from $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$, the randomness used in key generation and encryption. To obtain the key generation randomness from $\mathsf{SIM}_{\mathsf{NCE}}$, SIM must compile the set $\mathcal{Q}$ of all honestly generated ciphertexts and corresponding plaintexts (c, m, ℓ) that were encrypted under epk by any party (not just $\mathcal{P}$). The list of ciphertexts c and labels ℓ can be looked up in its own records $(\mathsf{encryptionrec}, sid, c, \bot_m, \ell, \bot_r, epk, \mathsf{false})$. Since $\mathcal{P}$ is now corrupt, SIM can obtain the corresponding plaintexts m by querying the decryption interface on $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ with $(\mathsf{DECRYPT}, sid, c, \ell)$. After thus composing the list $\mathcal{Q}$, SIM calls $\mathsf{SIM}_{\mathsf{NCE}}(\mathsf{keyleak}, \mathcal{Q})$ to obtain the key generation randomness r_{key}.

To obtain the randomness used for every encryption under epk performed by $\mathcal{P}$, SIM considers $\mathcal{P}$'s previous outputs from the encryption interface $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, epk)$ that it received from $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$. For each of these outputs, SIM calls $\mathsf{SIM}_{\mathsf{NCE}}$ with $(\mathsf{randomness}, (c, m, (\ell, sid)))$ to obtain the encryption randomness r. For ciphertexts generated by $\mathcal{P}$ under other public keys $pk \neq epk$, SIM also has records of the form $(\mathsf{encryptionrec}, sid, c, m, \ell, r, pk, \mathsf{false})$, so that SIM can simply give the stored r and update the record to $(\mathsf{encryptionrec}, sid, c, m, \ell, r, pk, \mathsf{true})$. The latter is important to ensure that randomness will not be given out for another party later. Namely, when encrypting under an adversarial public key pk, we cannot exclude that different randomness r, r' yield the same ciphertext c. If this happens for two encryption calls made by two different parties, and these parties later get corrupted, then the simulator must ensure that different randomness r and r' is given out to each of these parties. (Recall that the simulator doesn't learn which party initiated an encryption, so it doesn't know which randomness it used for which party.) By flagging a record with true when the randomness was given out, SIM ensures that the same randomness will not be given out again to a different party.

If any other party $\mathcal{Q}$ becomes corrupted, SIM needs to provide to $\mathcal{A}$ all $\mathcal{Q}$'s previous inputs and outputs, as well as the randomness used to create the ciphertexts that $\mathcal{Q}$ generated. For all previous encryption outputs $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, pk)$ to $\mathcal{Q}$, which SIM now obtains from $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$, $\mathsf{SIM}_{\mathsf{NCE}}$ is called with $(\mathsf{randomness}, (c, m, (\ell, sid)))$ to return the randomness r, which SIM simply passes on to $\mathcal{A}$. It also includes the randomnesses used for other public keys $pk \neq epk$, which are all stored in the records $(\mathsf{encryptionrec}, sid, c, m, \ell, r, pk, \mathsf{false})$.

b) Reduction From FULL-SIM: Fix an environment and a real-world adversary $\mathcal{A}$ that can distinguish the real protocol with $\mathcal{A}$ from the ideal functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ with simulator SIM. We can then construct an adversary $\mathcal{B}$ that breaks the FULL-SIM-security of the underlying encryption scheme with essentially the same probability (ignoring negligible parts).

Essentially, we let $\mathcal{B}$'s inputs and oracles handle the key generation, encryption, decryption, random-oracle, and corruption queries, much like SIM lets these be handled by $\mathsf{SIM}_{\mathsf{NCE}}$. First, $\mathcal{B}$ receives the public key epk from the challenger in the FULL-SIM experiment.

If the environment, before the KEYGEN interface is called, instructs an honest party to encrypt a message m by providing it with input $(\mathsf{ENCRYPT}, sid, pk, m, \ell)$, $\mathcal{B}$ simply calculates $(c; r) \xleftarrow{r} \mathsf{Enc}(pk, m, (\ell, sid))$, and outputs $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, pk)$. It also saves (c, m, ℓ, r, pk). If at any point this party becomes corrupted, the reduction simply hands over the corresponding rs stored in (c, m, ℓ, r, pk). Decryption queries are simply ignored before the KEYGEN interface is called. Queries to the random-oracle functionality for $sid' = (sid, \mathcal{F}_{\mathsf{RO}})$ are rewired to $\mathcal{B}$'s own RO oracle; for other session identifiers, it executes the real code of $\mathcal{F}_{\mathsf{RO}}$.

When at some point the environment instructs the honest key-pair owner $\mathcal{P}$ to generate its keys by providing input (KEYGEN, sid), then $\mathcal{B}$ embeds the key epk in the $(\mathsf{KEYCONF}, sid, epk)$ output. The encryption queries for a "incorrect" public keys $pk \neq epk$ are answered as before. However, for an encryption query with $pk = epk$, $\mathcal{B}$ refers to its own encryption oracle, by calling $\textsc{Enc}(m, (\ell, sid))$ which returns a ciphertext c. We then store $(c, m, \ell, \bot_r, epk)$ and output $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, epk)$. Decryption queries are handled similarly, i.e., on input $(\mathsf{DECRYPT}, sid, c, \ell)$, $\mathcal{B}$ calls $\textsc{Dec}(c, (\ell, sid))$ to obtain m and returns $(\mathsf{PLAINTEXT}, sid, c, m, \ell)$.

If at some point a party becomes corrupted, $\mathcal{B}$ needs to provide a consistent view of its state to $\mathcal{A}$. It uses the records $(c, m, \ell, \bot_r, epk)$ belonging to the corrupted party to call $\textsc{Rand}(c, (\ell, sid))$ to obtain the randomness r, which is passed to $\mathcal{A}$. The randomness used for "incorrect" public keys $pk \neq epk$ is obtained from its records of the form (c, m, ℓ, r, pk). If the key-pair holder becomes corrupted, $\mathcal{B}$ additionally calls $\textsc{GetSK}()$ to receive r_{key} and passes it to $\mathcal{A}$. At some point, the environment then outputs its guess b^* indicating whether it is running in the real world with $\mathcal{A}$ or in the ideal world with SIM. $\mathcal{B}$ returns b^* as its own guess, indicating that it is running in the FULL-SIM-real experiment with the real encryption scheme, or in FULL-SIM-ideal with $\mathsf{SIM}_{\mathsf{NCE}}$.

It is clear that if $\mathcal{B}$ is running in FULL-SIM-real, then the environment's view is exactly as when interacting with the real protocol. Likewise, if $\mathcal{B}$ is running in FULL-SIM-ideal, then all its oracle queries are handled by $\mathsf{SIM}_{\mathsf{NCE}}$, as is also done by the UC simulator SIM. We therefore have that $\mathcal{B}$'s advantage is negligibly close to that of $\mathcal{A}$. $\square$

We now know that any FULL-SIM-secure encryption scheme securely realizes $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$. To prove the equivalence, we also need to prove the other direction. In a nutshell, we show how one can build a FULL-SIM-secure scheme from $\mathcal{F}^{\mathcal{L}}_{\mathsf{SMT}}$, while any adversary against the resulting scheme can be used to construct an environment which can be used to distinguish between the real world and the ideal one.

Theorem 4 ($\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}} \implies \mathsf{FULL\text{-}SIM}$): Any protocol that securely realizes $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ gives rise to a FULL-SIM-secure non-committing encryption scheme.

Proof. The construction of the FULL-SIM-secure encryption scheme $\mathsf{NCE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ from an instantiation of $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ is quite straightforward. First, a choose random, yet correctly structured, sid. The key generation algorithm invokes the KEYGEN interface, and returns the resulting public key

epk as the public key. Encryption, and decryption, invoke the $\mathsf{ENCRYPT}$ and $\mathsf{DECRYPT}$ interfaces, respectively.

For the proof, we first construct a simulator $\mathsf{SIM}_{\mathsf{NCE}}$ for FULL-SIM-ideal given a UC-simulator SIM. Essentially, we have to implement the five oracles given in the FULL-SIM-experiment, by translating the answer given by SIM. Namely, for oracle RO, the queries received are simply re-routed to the SIM's random-oracle interface. epk is obtained by calling SIM with (KEYGEN, sid). SIM answers with $(\mathsf{KEYCONF}, sid, pk)$. This pk is the public key epk which is given to the adversary. For encryption (oracle $\textsc{Enc}$), SIM is called with $(\mathsf{ENCRYPTL}, sid, epk, \mathcal{L}(m), \ell)$ if oracle $\textsc{GetSK}$ has not been queried, or $(\mathsf{ENCRYPTM}, sid, epk, m, \ell)$ in all other cases. Here, (m, ℓ) is the corresponding input. In both cases, SIM responds with $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, epk)$. This c is given to the adversary. Decryption (oracle $\textsc{Dec}$) is similar, i.e., for each query (c, ℓ) to decrypt, and not seen so far, $(\mathsf{DECRYPT}, sid, c, \ell)$ is sent to SIM, which answers with $(\mathsf{PLAINTEXT}, sid, m)$. If (c, ℓ) has been seen before, $\mathsf{SIM}_{\mathsf{NCE}}$ directly returns m as given in the prior corresponding query to SIM. This m is simply given to the adversary. If the oracle $\textsc{Rand}$ on input (c, ℓ) is called, and (c, ℓ) has never been queried before, $\mathsf{SIM}_{\mathsf{NCE}}$ hands $(\mathsf{ENCRYPT}, sid, epk, m, \ell)$, and $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, pk)$ to SIM. SIM then responds with a complete execution history, where the used randomness r is contained. If (c, ℓ) has been seen before, $\mathsf{SIM}_{\mathsf{NCE}}$ looks up the prior corresponding answer from SIM, which contains r. In any case, the randomness r is then given to the adversary. Finally, the oracle $\textsc{GetSK}$ is simulated by handing over all lists of the form (KEYGEN, sid), $(\mathsf{KEYCONF}, sid, epk)$, $(\mathsf{ENCRYPTL}, sid, epk, \mathcal{L}(m), \ell)$, and $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, epk)$ to SIM, if this oracle has not been called before. In the execution history given to $\mathsf{SIM}_{\mathsf{NCE}}$, there is the randomness r_{key}, which can be returned to the adversary by $\mathsf{SIM}_{\mathsf{NCE}}$. If oracle $\textsc{GetSK}$ has been called before, $\mathsf{SIM}_{\mathsf{NCE}}$ directly hands over r_{key} to the adversary. Clearly, $\mathsf{SIM}_{\mathsf{NCE}}$ perfectly translates all requests.

In the second step, assume that there is an adversary $\mathcal{A}$ which wins against the constructed simulator $\mathsf{SIM}_{\mathsf{NCE}}$. We can then construct an environment $\mathcal{B}$, and an adversary $\mathcal{A}_d$ which together can distinguish between the ideal world with SIM, and the real world. Namely, we proceed as follows. $\mathcal{B}$ requests a public key epk from $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ by sending (KEYGEN, sid) to $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ (through the dummy party defined by sid). The received epk is embedded to initialize $\mathcal{A}$, which is run inside the adversary $\mathcal{A}_d$. Every query to the random oracle by $\mathcal{A}$ is also rewired. However, as we have local random oracles, we need a workaround. Namely, $\mathcal{B}$ directs $\mathcal{A}_d$ to query the random oracle for each query. The returned value is simply given to $\mathcal{A}$. On the ith encryption query for message m_i, $\mathcal{B}$ spawns a new party $\mathcal{P}_i$ which it feeds with $(\mathsf{ENCRYPT}, sid, epk, m_i, \ell_i)$. Upon receiving $(\mathsf{CIPHERTEXT}, sid, c_i, m_i, \ell_i, epk)$, $\mathcal{B}$ records (m_i, c_i, ℓ_i, i), and gives c_i to $\mathcal{A}$. Decryption queries are delegated through the functionality as well. The answer is simply given to $\mathcal{A}$. If $\mathcal{A}$ wants to see randomness of given ciphertext c_i with label ℓ_i, $\mathcal{B}$ looks up record (m_i, c_i, ℓ_i, i), and corrupts $\mathcal{P}_i$. The simulator then returns the randomness r_i somewhere in the execution history, which is then given to $\mathcal{A}$. Likewise, if $\mathcal{A}$ wants to see r_{key}, $\mathcal{B}$ corrupts the holder of the secret key (defined by sid), which presents the execution history, where r_{key} is stored. Eventually, whatever $\mathcal{A}$ outputs, is also output by $\mathcal{B}$. The probability that $\mathcal{B}$ wins its own game is the same as $\mathcal{A}$'s. ☐

VII. Universally Composable Signcryption

We illustrate the use of our non-committing encryption functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ by showing that the generic encrypt-then-sign approach that is known to be secure for building signcryption schemes under property-based definitions [50], [1] also works in the UC setting against adaptive adversaries, if one relies on strongly unforgeable signatures. Gjøsteen and Kråkmo [26] already considered UC-secure signcryption, but explicitly left security against adaptive corruptions as an open problem.

In a nutshell, signcryption is a primitive allowing a sender $\mathcal{S}$ to send an encrypted and authenticated message to a receiver $\mathcal{R}$, where both sender and receiver have their own key pairs. The goal of signcryption is to obtain better efficiency than a generic composition of encryption and signatures, but proving the generic construction secure is important to set a benchmark to which direct schemes can be compared. Interactive functionalities such as secure message transmission ($\mathcal{F}_{\mathsf{SMT}}$) are clearly not well suited to build a local primitive such as signcryption, so it is a good use case for our non-interactive functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$. Moreover, as our signcryption directly gives rise to a protocol implementing $\mathcal{F}_{\mathsf{SMT}}$, also this functionality cannot be instantiated in the standard model.

The signcryption functionality $\mathcal{F}^{\mathcal{L}}_{\mathsf{SC}}$ is depicted in Figure 9. Following Gjøsteen and Kråkmo [26], as well as our own design choices for $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$, we let the adversary determine the ciphertext strings. Unlike Gjøsteen and Kråkmo, however, our functionality includes labels and lets public keys be determined by the adversary, modeling the case where the public keys are not necessarily certified or registered through a PKI, which avoids implying a PKI.

a) Explanation: Let us give a high-level description what each interface does.

- $\mathsf{KeyGeneration}$. This interface allows the sender and receiver to generate their public keys. The key pair of the receiver is used to encrypt the message, while that of the sender will be used to authenticate it. For simplicity, we have merged the interface for both participants.
- $\mathsf{Signcryption}$. This interface allows the sender to create the signcryption of a message m to a receiver's public key pk_r. In case the public key is not the one registered for the receiver, or the receiver is corrupted, then the adversary learns the message. For the registered receiver's public key, the signcryption ciphertexts provided by the adversary must be unique.
- $\mathsf{De-signcryption}$. This interface allows the receiver to de-signcrypt a ciphertext. If the receiver was not initialized, this interface ignores requests. In case the signcryption was

1) `Key Generation`. Upon input (KEYGEN, sid) from a party $\mathcal{P}$:
   - If $sid \neq (\mathcal{P}, \cdot, sid')$ and $sid \neq (\cdot, \mathcal{P}, sid')$, ignore.
   - If there is a record $(\mathsf{keyrec}, sid, \mathcal{P}, \cdot)$, ignore.
   - Send $(\mathsf{KEYGEN}, sid, \mathcal{P})$ to $\mathcal{A}$ and wait for $(\mathsf{KEYCONF}, sid, \mathcal{P}, v)$ from $\mathcal{A}$.
   - If $v = \perp$, ignore.
   - Create record $(\mathsf{keyrec}, sid, \mathcal{P}, v)$.
   - Output $(\mathsf{KEYCONF}, sid, v)$ to $\mathcal{P}$.
2) `SignCryption`. On input $(\mathsf{SIGNCRYPT}, sid, m, \ell, pk_r)$ from party $\mathcal{P}$:
   - If $sid \neq (\mathcal{P}, \mathcal{Q}, sid')$, or $pk_r = \perp$, ignore.
   - If there is no record $(\mathsf{keyrec}, sid, \mathcal{P}, pk_s)$, ignore.
   - If there is a record $(\mathsf{keyrec}, sid, \mathcal{Q}, v)$, let $pk'_r \leftarrow v$.
   - If $pk_r = pk'_r$, and $\mathcal{Q}$ is honest, send $(\mathsf{SENDL}, sid, \mathcal{L}(m), \ell, pk_r)$ to $\mathcal{A}$ and wait for $(\mathsf{SIGNCRYPTRES}, sid, \ell, s)$ from $\mathcal{A}$.
   - Else, send $(\mathsf{SENDM}, sid, m, \ell, pk_r)$ to $\mathcal{A}$, and wait for $(\mathsf{SIGNCRYPTRES}, sid, \ell, s)$ from $\mathcal{A}$.
   - If there is a record $(\mathsf{sc}, sid, \cdot, \cdot, s, \cdot, pk_s, pk'_r)$, pk'_r taken from $(\mathsf{keyrec}, sid, \mathcal{Q}, pk'_r)$, where $sid = (\mathcal{P}, \mathcal{Q}, sid')$, ignore.
   - If $pk'_r = pk_r$, create record $(\mathsf{sc}, sid, m, \ell, s, \mathsf{true}, pk_s, pk_r)$.
   - Output $(\mathsf{SIGNCRYPTRES}, sid, m, \ell, s)$ to $\mathcal{P}$.
3) `De-signcryption`. On input $(\mathsf{DESIGNCRYPT}, sid, s, \ell, pk_s)$ from party $\mathcal{P}$:
   - If $sid \neq (\mathcal{Q}, \mathcal{P}, sid')$, or $pk_s = \perp$, ignore.
   - If there is no record $(\mathsf{keyrec}, sid, \mathcal{P}, pk_r)$, ignore.
   - If there is a record $(\mathsf{sc}, sid, m, \ell, s, b, pk_s, pk_r)$, output $(\mathsf{DESIGNCRYPT}, sid, s, m, \ell, b)$ to $\mathcal{P}$.
   - If there is a record $(\mathsf{keyrec}, sid, \mathcal{Q}, pk_s)$, and $\mathcal{Q}$ is honest, create record $(\mathsf{sc}, sid, m, \ell, s, \mathsf{false}, pk_s, pk_r)$, and output $(\mathsf{DESIGNCRYPT}, sid, s, m, \ell, \mathsf{false})$ to $\mathcal{P}$.
   - Else, send $(\mathsf{DESIGNCRYPT}, sid, s, \ell, pk_s)$ to $\mathcal{A}$ and wait for $(\mathsf{DESIGNCRYPTA}, sid, s, m, \ell, \phi)$ from $\mathcal{A}$.
   - If $\phi \notin \{\mathsf{false}, \mathsf{true}\}$, ignore.
   - If there is no record $(\mathsf{sc}, sid, m, \ell, \cdot, \mathsf{true}, pk'_s, pk_r)$, and $pk'_s = pk_s$, where pk'_s is taken from $(\mathsf{keyrec}, sid, \mathcal{Q}, pk'_s)$, let $\phi \leftarrow \mathsf{false}$.
   - If $\phi = \mathsf{false}$, let $m \leftarrow \perp$.
   - Create record $(\mathsf{sc}, sid, m, \ell, s, \phi, pk_s, pk_r)$.
   - Output $(\mathsf{DESIGNCRYPT}, sid, s, m, \ell, \phi)$ to $\mathcal{P}$.

Fig. 9. Our functionality $\mathcal{F}_{\mathsf{SC}}^{\mathcal{L}}$ with leakage function $\mathcal{L}$.

honestly generated, the functionality can directly answer the requests. In all other cases, the adversary has to determine whether the ciphertext is valid and, if so, provide the plaintext. The adversary is then committed to its decision, in the sense that future de-signcryptions of the same ciphertext will yield the same result.

b) Protocol Description: We now describe the generic encrypt-then-sign construction for a signcryption scheme using $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and $\mathcal{F}_{\mathsf{Sig}}$ as sub-functionalities. $\mathcal{F}_{\mathsf{Sig}}$ is the ideal signature functionality, enforcing strong unforgeability. The definition of $\mathcal{F}_{\mathsf{Sig}}$ can be found in the full version of this paper.

Step 1a. Key Generation (Sender $\mathcal{S}$):

1) Upon input (KEYGEN, sid), ignore, if a record $(\mathsf{key}, sid, \cdot)$ exists.
2) If $sid \neq (\mathcal{S}, \mathcal{R}, sid')$, ignore.
3) Create a signature key pair by inputting $(\mathsf{KEYGEN}, (\mathcal{S}, (sid, \mathcal{F}_{\mathsf{Sig}})))$ to $\mathcal{F}_{\mathsf{Sig}}$.
4) Upon obtaining output $(\mathsf{KEYCONF}, (\mathcal{S}, (sid, \mathcal{F}_{\mathsf{Sig}})), spk)$, create a record (key, sid, spk).
5) Output $(\mathsf{KEYCONF}, sid, spk)$.

Step 1b. Key Generation (Receiver $\mathcal{R}$):

1) Upon input (KEYGEN, sid), ignore, if a record $(\mathsf{key}, sid, \cdot)$ exists.
2) If $sid \neq (\mathcal{S}, \mathcal{R}, sid')$, ignore.
3) Create an encryption scheme key pair by inputting $(\mathsf{KEYGEN}, (\mathcal{R}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})))$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$.
4) Upon obtaining output $(\mathsf{KEYCONF}, (\mathcal{R}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), epk)$, create a record (key, sid, epk).
5) Output $(\mathsf{KEYCONF}, sid, epk)$.

Step 2. Create Signcryption (Sender $\mathcal{S}$):

1) Upon input $(\mathsf{SIGNCRYPT}, sid, m, \ell, pk)$, ignore, if no record (key, sid, spk) exists.
2) If $sid \neq (\mathcal{S}, \mathcal{R}, sid')$ or $pk = \perp$, ignore.
3) Encrypt m under pk with label $\ell' = (\ell, spk, pk)$, i.e., input $(\mathsf{ENCRYPT}, (\mathcal{R}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), pk, m, (\ell, spk, pk))$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$.
4) Sign the resulting ciphertext together with the label ℓ', i.e., upon obtaining $(\mathsf{CIPHERTEXT}, (\mathcal{R}, (sid, \mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}})), c, m, (\ell, spk, pk), pk)$ from $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, send $(\mathsf{SIGN}, (\mathcal{S}, sid), (c, (\ell, spk, pk)))$ to $\mathcal{F}_{\mathsf{Sig}}$. Upon receiving $(\mathsf{SIGNATURE}, (\mathcal{S}, sid), (c, (\ell, spk, pk)), \sigma)$, output $(\mathsf{SIGNCRYPTRES}, sid, m, \ell, (c, \sigma))$.

Step 3. De-signcryption (Receiver $\mathcal{R}$):

1) Upon receiving $(\mathsf{DESIGNCRYPT}, sid, s, \ell, pk)$, ignore if there is no record (key, sid, epk) or $pk = \perp$.
2) If $s \neq (c, \sigma)$ or $sid \neq (\mathcal{S}, \mathcal{R}, sid')$, ignore.
3) Verify the signature, i.e., input $(\mathsf{VERIFY}, (\mathcal{S}, sid), (c, (\ell, pk, epk)), \sigma, pk)$ to $\mathcal{F}_{\mathsf{Sig}}$. Upon receiving $(\mathsf{VERIFY}, (\mathcal{S}, sid), (m, (\ell, pk, epk)), \sigma, pk, v)$ from $\mathcal{F}_{\mathsf{Sig}}$, output $(\mathsf{DESIGNCRYPT}, sid, s, \perp, (\ell, pk, epk), \mathsf{false})$, if $v = \mathsf{false}$. Else, decrypt the ciphertext, i.e., send $(\mathsf{DECRYPT}, (\mathcal{R}, sid), c, (\ell, pk, epk))$ to $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$. Upon receiving $(\mathsf{PLAINTEXT}, (\mathcal{R}, sid), c, m, (\ell, pk, epk))$, output $(\mathsf{DESIGNCRYPT}, sid, s, m, \ell, \mathsf{true})$.

The full proof of the following theorem is given in Appendix C.

Theorem 5: The above construction securely realizes $\mathcal{F}_{\mathsf{SC}}^{\mathcal{L}}$ in the $(\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}, \mathcal{F}_{\mathsf{Sig}})$-hybrid model with adaptive corruptions without secure erasures.

Proof-Sketch. The proof is straightforward. Namely, signatures are simulated honestly, while ciphertexts are programmed as soon as any party becomes corrupted.

We want to stress that defining a functionality for multiple senders, and receivers, is straightforward.

VIII. Conclusion

We defined and gave a secure realization of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$, a UC functionality for local public-key encryption, in a setting of adaptive corruptions without erasures. We also introduced a new property-based notion of FULL-SIM security, which we showed to be equivalent to our UC definition. Our non-interactive construction assumes the existence of trapdoor one-way permutations in the random-oracle model. It is very efficient, as it only has a constant overhead, only requires one evaluation of a trapdoor permutation, and a handful of hash evaluations. Due to Nielsen's impossibility result for round-optimal adaptively secure secure message transmission, a standard model instantiation of our primitive is also impossible.

Still, we believe that our functionality $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and its instantiation can be a very useful building block for other UC-secure protocols, which until now had to provide direct constructions tailored to one specific application scenario. As an example, we showed how $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ can be used to model and realize the signcryption functionality $\mathcal{F}_{\mathsf{SC}}^{\mathcal{L}}$ in the presence of adaptive adversaries without erasures.

IX. Acknowledgements

The authors were supported by the European Research Council under grant agreement number 321310 (PERCY).

References

[1] An, J.H., Dodis, Y., Rabin, T.: On the security of joint signature and encryption. In: Eurocrypt. pp. 83–107 (2002)

[2] Beaver, D.: Plug and play encryption. In: Crypto. pp. 75–89 (1997)

[3] Beaver, D., Haber, S.: Cryptographic protocols provably secure against dynamic adversaries. In: Eurocrypt. pp. 307–323 (1992)

[4] Bellare, M., Dowsley, R., Waters, B., Yilek, S.: Standard security does not imply security against selective-opening. In: Eurocrypt. pp. 645–662 (2012)

[5] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: CCS. pp. 62–73 (1993)

[6] Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: Eurocrypt. pp. 92–111 (1994)

[7] Böhl, F., Hofheinz, D., Kraschewski, D.: On definitions of selective opening security. In: PKC. pp. 522–539 (2012)

[8] Boldyreva, A., Fischlin, M.: Analysis of random oracle instantiation scenarios for OAEP and other practical schemes. In: CRYPTO. pp. 412–429 (2005)

[9] Camenisch, J., Enderlein, R.R., Krenn, S., Küsters, R., Rausch, D.: Universal composition with responsive environments. ePrint 34 (2016)

[10] Camenisch, J., Enderlein, R.R., Neven, G.: Two-server password-authenticated secret sharing uc-secure against transient corruptions. In: PKC. pp. 283–307 (2015)

[11] Camenisch, J., Lehmann, A., Neven, G., Samelin, K.: Virtual smart cards: How to sign with a password and a server. In: SCN. pp. 353–371 (2016)

[12] Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Crypto. pp. 126–144 (2003)

[13] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. ePrint 67 (2000)

[14] Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multi-party computation. In: STOC. pp. 639–648 (1996)

[15] Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. J. ACM 51(4), 557–594 (2004)

[16] Canetti, R., Halevi, S., Katz, J.: Adaptively-secure, non-interactive public-key encryption. In: TCC. pp. 150–168 (2005)

[17] Canetti, R., Herzog, J.: Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In: TCC. pp. 380–403 (2006)

[18] Canetti, R., Jain, A., Scafuro, A.: Practical UC security with a global random oracle. In: CCS. pp. 597–608 (2014)

[19] Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing chosen-ciphertext security. In: CRYPTO. pp. 565–582 (2003)

[20] Choi, S.G., Dachman-Soled, D., Malkin, T., Wee, H.: Improved non-committing encryption with applications to adaptively secure protocols. In: Asiacrypt. pp. 287–302 (2009)

[21] Damgård, I., Nielsen, J.B.: Improved non-committing encryption schemes based on a general complexity assumption. In: Crypto. pp. 432–450 (2000)

[22] Dwork, C., Naor, M., Reingold, O., Stockmeyer, L.J.: Magic functions. In: FOCS. pp. 523–534 (1999)

[23] Fehr, S., Hofheinz, D., Kiltz, E., Wee, H.: Encryption schemes secure against chosen-ciphertext selective opening attacks. In: Eurocrypt. pp. 381–402 (2010)

[24] Fuchsbauer, G., Heuer, F., Kiltz, E., Pietrzak, K.: Standard security does imply security against selective opening for markov distributions. In: TCC-1. pp. 282–305 (2016-A)

[25] Garay, J.A., Wichs, D., Zhou, H.: Somewhat non-committing encryption and efficient adaptively secure oblivious transfer. In: Crypto. pp. 505–523 (2009)

[26] Gjøsteen, K., Kråkmo, L.: Universally composable signcryption. In: EuroPKI. pp. 346–353 (2007)

[27] Goldwasser, S., Kalai, Y.T.: On the (in)security of the fiat-shamir paradigm. In: FOCS. pp. 102–113 (2003)

[28] Hazay, C., Lindell, Y., Patra, A.: Adaptively secure computation with partial erasures. In: PODC. pp. 291–300 (2015)

[29] Hazay, C., Patra, A.: Efficient one-sided adaptively secure computation. J. Cryptology 30(1), 321–371 (2017)

[30] Hazay, C., Patra, A., Warinschi, B.: Selective opening security for receivers. ePrint 860 (2015)

[31] Hemenway, B., Ostrovsky, R., Richelson, S., Rosen, A.: Adaptive security with quasi-optimal rate. In: TCC-A I. pp. 525–541 (2016)

[32] Hemenway, B., Ostrovsky, R., Rosen, A.: Non-committing encryption from Φ-hiding. In: TCC. pp. 591–608 (2015)

[33] Heuer, F., Poettering, B.: Selective opening security from simulatable data encapsulation. IACR Cryptology ePrint Archive 2016, 845 (2016)

[34] Hofheinz, D.: Possibility and impossibility results for selective decommitments. J. Cr. 24(3), 470–516 (2011)

[35] Hofheinz, D., Jager, T., Rupp, A.: Public-key encryption with simulation-based selective-opening security and compact ciphertexts. ePrint 180 (2016)

[36] Hofheinz, D., Rao, V., Wichs, D.: Standard security does not imply indistinguishability under selective opening. ePrint 792 (2015)

[37] Hofheinz, D., Rupp, A.: Standard versus selective opening security: Separation and equivalence results. In: TCC. pp. 591–615 (2014)

[38] Jarecki, S., Lysyanskaya, A.: Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In: Eurocrypt. pp. 221–242 (2000)

[39] Küsters, R., Tuengerthal, M.: Joint state theorems for public-key encryption and digital signature functionalities with local computation. In: CSF. pp. 270–284 (2008)

[40] Küsters, R., Tuengerthal, M.: Universally composable symmetric encryption. In: CSF. pp. 293–307 (2009)

[41] Küsters, R., Tuengerthal, M.: Ideal key derivation and encryption in simulation-based security. In: CT-RSA. pp. 161–179 (2011)

[42] Lei, F., Chen, W., Chen, K.: A non-committing encryption scheme based on quadratic residue. In: ISCIS. pp. 972–980 (2006)

[43] Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: STOC. pp. 427–437 (1990)

[44] Nielsen, J.B.: Non-committing encryption is too easy in the random oracle model. Tech. rep., BRICS (2001)

[45] Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In: Crypto. pp. 111–126 (2002)

[46] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT. pp. 223–238 (1999)

[47] Paillier, P., Vergnaud, D.: Discrete-log-based signatures may not be equivalent to discrete log. In: ASIACRYPT. pp. 1–20 (2005)

[48] Schnorr, C.: Efficient signature generation by smart cards. J. Cryptology 4(3), 161–174 (1991)

[49] Shoup, V., Gennaro, R.: Securing threshold cryptosystems against chosen ciphertext attack. J. Cryptology 15(2), 75–96 (2002)

[50] Zheng, Y.: Digital signcryption or how to achieve cost(signature & encryption) $<<$ cost(signature) + cost(encryption). In: Crypto. pp. 165–179 (1997)

[51] Zhu, H., Bao, F.: Error-free, multi-bit non-committing encryption with constant round complexity. In: Inscrypt. pp. 52–61 (2010)

Appendix A
Nielsen's Constructions

The construction of the underlying encryption scheme by Nielsen looks very similar to ours, but without labels and without $c_{(3)}$. We omit the message and party IDs to focus on the construction itself, which essentially the IND-CPA-secure Bellare-Rogaway encryption-scheme [5].

1) `Send`. Upon input $(\mathsf{SEND}, mid, j, m)$ of party $\mathcal{P}_i$ do:

- Deliver $(\mathsf{RECEIVE}, mid, i, m)$ to $\mathcal{P}_j$, and send $(\mathsf{RECEIVE}, mid, i, \mathcal{L}(m))$ to $\mathcal{A}$.

Fig. 10. Secure Message Transmission functionality [44].

$\mathsf{KGen}(1^\tau)$: Generate a random trapdoor one-way permutation, i.e., $(f, f^{-1}, \Sigma) \xleftarrow{r} \mathsf{TDP.Gen}(1^\tau)$. Set $\mathcal{M} = \{0,1\}^\tau$. Output the public key $epk = (f, \Sigma)$, and $esk = f^{-1}$ as the secret key.

$\mathsf{Enc}(epk, m)$: Draw $x \xleftarrow{r} \mathsf{TDP.Sample}_\Sigma(1^\tau)$ (Σ is given by epk), and compute $c_{(1)} \leftarrow \mathsf{TDP}.f(x)$, and $c_2 \leftarrow \mathcal{H}(x) \oplus m$. Output the ciphertext $c \leftarrow (c_{(1)}, c_{(2)})$.

$\mathsf{Dec}(esk, c)$: Parse c as $(c_{(1)}, c_{(2)})$. Compute $x' \leftarrow \mathsf{TDP}.f^{-1}(c_{(1)})$, and $m' \leftarrow \mathcal{H}(x') \oplus c_{(2)}$. Output m'.

A. Nielsen's Secure Message Transfer

As usual, $\mathcal{L}$ returns the length of a message m. Depicted in Figure 10 is the plain definition given by Nielsen [44].

Appendix B
Proof of Propositions 1-5

A. Proof of Proposition 1

Proof. Assume there is an adversary $\mathcal{A}$ which can guess whether it is run in RECV-SIM-real, or RECV-SIM-ideal resp., with a probability non-negligibly better than $\frac{1}{2}$. We can then construct an adversary $\mathcal{B}$ which uses $\mathcal{A}$ internally to distinguish between FULL-SIM-real and FULL-SIM-ideal with the same probability.

$\mathcal{B}$ proceeds as follows. It receives epk from its own challenger. It then initializes $\mathcal{A}$ with epk. Then, for every query s to $\mathcal{H}$ from $\mathcal{A}$, $\mathcal{B}$ forwards the query to its own oracle $\mathcal{H}$, and returns the result unmodified to $\mathcal{A}$. Likewise, for every query (m, ℓ) to Enc, $\mathcal{B}$ asks the encryption oracle provided by its own challenger. Again, the answer is passed to $\mathcal{A}$ unmodified. Finally, every query (c, ℓ) to the decryption is also honestly answered using the decryption oracle provided to $\mathcal{B}$. Eventually, $\mathcal{A}$ returns $\mathsf{state}_\mathcal{A}$. $\mathcal{B}$ saves $\mathsf{state}_\mathcal{A}$. $\mathcal{B}$ then asks its own oracle $\mathsf{GetSK}()$ to receive r_{key}. $\mathcal{B}$ then generates the corresponding secret key of epk by letting $(epk', esk) \xleftarrow{r} \mathsf{KGen}(1^\tau; r_{\mathsf{key}})$. $\mathcal{B}$ then continues simulating $\mathcal{A}$ with input $(esk, \mathsf{state}_\mathcal{A})$. The random oracle queries made by $\mathcal{A}$ are answered as before. Finally, $\mathcal{A}$ will return its guess b^*. $\mathcal{B}$ uses b^* as its own guess. Clearly, $\mathcal{B}$'s advantage is the same as $\mathcal{A}$'s, as $\mathcal{B}$ can perfectly simulate $\mathcal{A}$'s environment. $\square$

B. Proof of Proposition 2

Proof. Let $\mathcal{A}$ be an adversary, together with a distinguisher D, which together guess in which RSIM-SO-experiment it is run in with a probability non-negligibly better than $\frac{1}{2}$. We can then construct an adversary $\mathcal{B}$ which can distinguish between FULL-SIM-real and FULL-SIM-ideal with a non-negligible probability.

We prove this statement by a series of hybrids. Let Exp_0 be RSIM-SO-ideal, while Exp_n is the experiment RSIM-SO-real. Also, define Exp_i such that the first i public keys are simulated, while the remaining $n - i$ are honestly generated. Further assume towards contradiction that there is an adversary $\mathcal{A}$ that can distinguish between Exp_i, and Exp_{i+1} for some index i. We can then construct an adversary $\mathcal{B}$ which can break our FULL-SIM-security definition. In particular, $\mathcal{B}$ proceeds as follows. It receives epk from its own challenger. It then embeds epk as epk_{i+1}, and leaves the other epks untouched, and gives the complete public key vector to $\mathcal{A}$. Algorithm $\mathcal{B}$ then samples the message vector $\mathbf{m}$ according to the received distribution Dist. It generates each c_j, $j \neq i$ according to the current experiment, but asks its own challenger to receive the ciphertext c_i. The ciphertext vector $\mathbf{c}$ is then given to $\mathcal{A}$. Eventually, $\mathcal{A}$ returns $\mathcal{I}$, and $\mathcal{B}$ then receives all the randomness used to create the corresponding secret keys, and the messages $\mathbf{m}_\mathcal{I}$, which it provides to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs output, which $\mathcal{B}$, together with $\mathbf{m}$, and $\mathcal{I}$ to D. Whatever D then outputs, is then also output by $\mathcal{B}$. Clearly, the probability that $\mathcal{B}$ can successfully distinguish between FULL-SIM-real, and FULL-SIM-ideal is thus non-negligible. $\square$

C. Proof of Proposition 3

Proof. Assume we have a distinguisher D with some arbitrary, but fixed adversary $\mathcal{A}$ which together can decide whether they run in SSIM-SO-real or SSIM-SO-ideal with a probability non-negligibly better than $\frac{1}{2}$. We can then construct an adversary $\mathcal{B}$ which distinguishes between FULL-SIM-real and FULL-SIM-ideal with the same probability.

In the first step, the random oracle is rewired to $\mathcal{B}$'s own random oracle. $\mathcal{B}$ then receives the challenge public key epk. epk is simply passed to $\mathcal{A}$ to initialize the adversary. $\mathcal{B}$ receives Dist and state_1 from $\mathcal{A}$. $\mathcal{B}$ then samples the message vector $\mathbf{m}$ according to the received Dist. Each m_i is then sent to $\mathcal{B}$'s own encryption oracle to receive each ciphertext c_i. Let $\mathbf{c}$ denote the complete vector of the ciphertexts. $\mathcal{A}$ is then given state_1 and $\mathbf{c}$. Eventually, $\mathcal{A}$ outputs $(\mathcal{I}, \mathsf{state}_2)$. For each $i \in \mathcal{I}$, $\mathcal{B}$ then queries its own oracle $\mathsf{Rand}(\cdot)$ with c_i to receive each r_i. Let the vector of all received r_i be $\mathbf{r}$. $\mathcal{A}$ is then given $(\mathbf{r}, \mathbf{m}_\mathcal{I}, \mathsf{state}_2)$. Finally, $\mathcal{A}$ returns output. D is given $(\mathbf{m}, \mathsf{Dist}, \mathcal{I}, \mathsf{output})$. Whatever D outputs, is also output by $\mathcal{B}$. Clearly, as we can perfectly simulate the environment of $\mathcal{A}$ and D, $\mathcal{B}$ success probability equals the one of $\mathcal{A}$ with D. $\square$

D. Proof of Proposition 4

Proof. For the first direction, note that there are constructions in the standard model, i.e., under the DCR-Assumption, for IND-NCER security [16], but none for FULL-SIM security, as proven before. Thus, IND-NCER security does not imply FULL-SIM security.

For the other direction, we show that there is a FULL-SIM-secure construction, which is not IND-NCER-secure. Namely, we already know that our construction is FULL-SIM-secure, if TDPs exist in the random oracle model. We now show that our construction is not IND-NCER-secure.

Let Open, and Enc^* be arbitrarily defined, while the remaining algorithms are defined as in our construction. In particular, $\mathcal{A}$ draws a random message $m \xleftarrow{r} \{0,1\}^\tau$. Then, the challenger (Note, Open and Enc^* are public, i.e., there is no random-oracle programming possible) needs to return a ciphertext c, which correctly decrypts to m, i.e., at least $c_3 = \mathcal{H}(x, k, m, \ell)$ must hold. The probability that Enc^* guesses the messages correctly upfront (and therefore the correct output) is negligible, i.e., at most $q_h/2^\tau$, where q_h is the number of random oracle queries. The other case is similar, i.e., that c_3 was drawn randomly, and one hopes that the unique random oracle query (x, k, m, ℓ) (Note, k,ℓ, and x are fixed), makes the ciphertext valid. Clearly, this is negligible as well. Thus, the probability that Open returns randomness for the secret key such that the ciphertext returned decrypts correctly is negligible ($1/2^\tau$), regardless of the choice of Open, and Enc^*. It thus follows that the probability that this happens is equal/less than $(q_h + 1)/2^\tau$, which is negligible. □

E. Proof of Proposition 5

Proof. The idea of the proof is as follows. If there exists is a perfectly-binding (bit-)commitment-scheme, then RECV-SIM-Security does not imply FULL-SIM-Security. With perfectly-binding we mean that even a computationally unbounded adversary can find only one way to open a given commitment c w.r.t. to the (even adversarially chosen) public parameters pp. Refer to the full version of this paper for additional information.

Let NCE be any FULL-SIM-secure encryption, and COMMIT be a perfectly-binding commitment-scheme as defined before. We now construct NCE' as follows, such that is only RECV-SIM-secure, but not FULL-SIM-secure.

$\mathsf{NCE}'.\mathsf{KGen}(1^\tau)$: Generate $(epk, esk) \xleftarrow{r} \mathsf{NCE}.\mathsf{KGen}(1^\tau)$. Return (epk, esk).

$\mathsf{NCE}'.\mathsf{Enc}(epk, m, \ell)$: Generate $pp \xleftarrow{r} \mathsf{COMMIT}.\mathsf{PPGen}(1^\tau)$. Then, for each bit $b_i \in m$, let $(c_i, r_i) \xleftarrow{r} \mathsf{COMMIT}.\mathsf{Commit}(pp, b_i)$. Set $\ell' \leftarrow (\ell, pp, (c_1, c_2, \ldots, c_{|m|}))$. Let $c \xleftarrow{r} \mathsf{NCE}.\mathsf{Enc}(epk, m, \ell')$, and return $(c, pp, (c_1, c_2, \ldots, c_{|m|}))$.

$\mathsf{NCE}'.\mathsf{Dec}(esk, c, \ell)$: Parse c as $(c, pp, (c_1, c_2, \ldots, c_{|m|}))$. Let $\ell' \leftarrow (\ell, pp, (c_1, c_2, \ldots, c_{|m|}))$ Return $\mathsf{NCE}.\mathsf{Dec}(esk, c, \ell')$.

It is obvious that the construction is still RECV-SIM-secure by the following argument. If the commitment scheme used is computationally hiding, the ciphertext c is indistinguishable from an encryption using any other message m' of the same length, as the randomnesses used to generate the commitments are never given to the adversary attacking the scheme. Thus, the simulator can choose a random message of the same length.

However, the scheme NCE' cannot be FULL-SIM-secure, as the probability that any simulator $\mathsf{SIM}_{\mathsf{NCE}}$ guesses the correct message is negligible, as the commitment-scheme is perfectly binding, i.e., no simulator can equivocate the commitments. For any other (meaningful) definition of the leakage oracle $\mathcal{L}$, similar arguments exist. □

Appendix C
Proof of Theorem 5

Proof. Given a $(\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}, \mathcal{F}_{\mathsf{Sig}})$-hybrid-model adversary $\mathcal{A}$, we construct a simulator SIM so that no environment can distinguish running with $\mathcal{A}$ and the real protocol in the $(\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}, \mathcal{F}_{\mathsf{Sig}})$-hybrid world from running with SIM and $\mathcal{F}^{\mathcal{L}}_{\mathsf{SC}}$ in the ideal world. Note that, apart from interacting correctly with $\mathcal{F}^{\mathcal{L}}_{\mathsf{SC}}$, the simulator must also play the role of the subfunctionalities $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$ and $\mathcal{F}_{\mathsf{Sig}}$ to $\mathcal{A}$ when they are called by corrupt parties. The simulator SIM proceeds as follows:

Key Generation (Sender). Upon receiving $(\mathsf{KEYGEN}, sid, \mathcal{S})$ from $\mathcal{F}^{\mathcal{L}}_{\mathsf{SC}}$, send $(\mathsf{KEYGEN}, (\mathcal{S}, (sid, \mathcal{F}_{\mathsf{Sig}}))))$ to $\mathcal{A}$ and wait for $(\mathsf{KEYCONF}, (\mathcal{S}, (sid, \mathcal{F}_{\mathsf{Sig}}))), spk)$ from $\mathcal{A}$. Create a record $(\mathsf{keyrec}, sid, \mathcal{S}, spk)$ and send $(\mathsf{KEYCONF}, sid, \mathcal{S}, spk)$ to $\mathcal{F}^{\mathcal{L}}_{\mathsf{SC}}$.

Key Generation (Receiver). Upon receiving $(\mathsf{KEYGEN}, sid, \mathcal{R})$ from $\mathcal{F}^{\mathcal{L}}_{\mathsf{SC}}$, send $(\mathsf{KEYGEN}, (\mathcal{R}, (sid, \mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}})))$ to $\mathcal{A}$ and wait for $(\mathsf{KEYCONF}, (\mathcal{R}, (sid, \mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}))), epk)$ from $\mathcal{A}$. Create a record $(\mathsf{keyrec}, sid, \mathcal{R}, epk)$ and send $(\mathsf{KEYCONF}, sid, \mathcal{R}, epk)$ to $\mathcal{F}^{\mathcal{L}}_{\mathsf{SC}}$.

Encryption. Upon receiving $(\mathsf{ENCRYPT}, sid', epk', m, \ell)$ from $\mathcal{A}$, SIM executes the real code of $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$, asking $\mathcal{A}$ to provide a ciphertext c, creating an encryption record $(\mathsf{encryptionrec}, sid, epk', m, \ell, c)$, and returning $(\mathsf{CIPHERTEXT}, sid, c, m, \ell, epk')$.

Decryption. Upon receiving $(\mathsf{DECRYPT}, sid', c, \ell)$ from $\mathcal{A}$ in name of the corrupt receiver $\mathcal{R}$, SIM executes the real code of $\mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}$, asking $\mathcal{A}$ for a plaintext m if necessary, creating a decryption record $(\mathsf{decryptionrec}, sid, m, \ell, c)$, and returning $(\mathsf{PLAINTEXT}, sid, c, m, \ell)$.

Signing. Upon receiving (SIGN, sid', m) from $\mathcal{A}$ in name of the corrupt sender $\mathcal{S}$, SIM executes the real code of $\mathcal{F}_{\mathsf{Sig}}$, asking $\mathcal{A}$ for a signature σ, creating a signing record $(\mathsf{signature}, sid, m, \sigma, spk, \mathsf{true})$, and returning $(\mathsf{SIGNATURE}, sid, m, \sigma)$.

Verification. Upon receiving $(\mathsf{VERIFY}, sid', m, \sigma, spk')$ from $\mathcal{A}$, SIM executes the real code of $\mathcal{F}_{\mathsf{Sig}}$, asking $\mathcal{A}$ for a verification outcome ϕ if necessary, creating signature record $(\mathsf{signature}, sid', m, \cdot, spk', \mathsf{true})$ or $(\mathsf{signature}, sid', m, \sigma, spk', \phi)$, and returning $(\mathsf{VERIFY}, sid', m, \sigma, spk', \phi)$.

Signcryption. When $\mathcal{S}$ and $\mathcal{R}$ are both honest and $pk_r = epk$, SIM is notified by $\mathcal{F}^{\mathcal{L}}_{\mathsf{SC}}$ by receiving $(\mathsf{SENDL}, sid, \mathcal{L}(m), \ell, pk_r)$. The simulator doesn't know the message m, but obtains a ciphertext by sending $(\mathsf{ENCRYPTL}, (sid, \mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}), epk, \mathcal{L}(m), (\ell, spk, epk))$ to $\mathcal{A}$ and waiting for $(\mathsf{CIPHERTEXT}, (sid, \mathcal{F}^{\mathcal{L}}_{\mathsf{NCE}}), c)$ from $\mathcal{A}$. It then creates an incomplete encryption record $(\mathsf{encryptionrec}, sid, epk, \perp, (\ell, spk, epk), c)$, indicating that it doesn't know the corresponding message m. It then obtains a signature σ for (c, ℓ) using the procedure for signing simulation above and sends $(\mathsf{SIGNCRYPTRES}, sid, (\ell, spk, epk), s = (c, s))$ back to $\mathcal{F}^{\mathcal{L}}_{\mathsf{SC}}$.

When $\mathcal{S}$ is honest and ($\mathcal{R}$ is corrupt or $pk_r \neq epk$), then SIM executes the real signcryption algorithm with the code of $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and $\mathcal{F}_{\mathsf{Sig}}$.

De-signcryption. When a new de-signcryption request occurs, SIM is notified by $\mathcal{F}_{\mathsf{SC}}^{\mathcal{L}}$ by a message $(\mathsf{DESIGNCRYPT}, sid, s = (c, \sigma), \ell, pk_s)$. First check whether the signature σ should be deemed valid for message $(c, (\ell, pk_s, epk))$ by performing the verification simulation above. If not, then SIM sends $(\mathsf{DESIGNCRYPTA}, sid, s, \perp, \ell, \mathsf{false})$ back to $\mathcal{F}_{\mathsf{SC}}^{\mathcal{L}}$, indicating that the signcryption is invalid. If so, then it must be the case that $pk_s \neq spk$, because the strong unforgeability enforced by $\mathcal{F}_{\mathsf{SC}}^{\mathcal{L}}$ would have rejected the ciphertext already. If c was part of a signcryption generated by the honest sender (for which we don't know the corresponding plaintext), then SIM sends $(\mathsf{DESIGNCRYPTA}, sid, s, \perp, \ell, \mathsf{false})$ back to $\mathcal{F}_{\mathsf{SC}}^{\mathcal{L}}$, because the ciphertext label of c includes the wrong sender's public key $spk \neq pk_s$. Otherwise, $\mathcal{A}$ decrypts c by performing the decryption simulation above to obtain the plaintext message m. If $m = \perp$, then send $(\mathsf{DESIGNCRYPTA}, sid, s, \perp, \ell, \mathsf{false})$ back to $\mathcal{F}_{\mathsf{SC}}^{\mathcal{L}}$, else send $(\mathsf{DESIGNCRYPTA}, sid, s, m, \ell, \mathsf{true})$ back to $\mathcal{F}_{\mathsf{SC}}^{\mathcal{L}}$.

Corruption. When a party is corrupted, then SIM obtains the full input and output history of that party. Based on this history, SIM can also compile the list of inputs and outputs of that party to the $\mathcal{F}_{\mathsf{NCE}}^{\mathcal{L}}$ and $\mathcal{F}_{\mathsf{Sig}}$ sub-functionalities, which it couldn't do earlier because some of the messages were unknown. It submits this full list of inputs and outputs to $\mathcal{A}$.

$\square$