Adaptively-Secure, Non-Interactive Public-Key Encryption*

Ran Canetti† canetti@watson.ibm.com

Shai Halevi† shaih@alum.mit.edu

Jonathan Katz‡ jkatz@cs.umd.edu

November 23, 2004

Abstract

Adaptively-secure encryption schemes ensure secrecy even in the presence of an adversary who can corrupt parties in an adaptive manner based on public keys, ciphertexts, and secret data of already-corrupted parties. Ideally, an adaptively-secure encryption scheme should, like standard public-key encryption, allow arbitrarily-many parties to use a single encryption key to securely encrypt arbitrarily-many messages to a given receiver who maintains only a single short decryption key. However, it is known that these requirements are impossible to achieve: no non-interactive encryption scheme that supports encryption of an unbounded number of messages and uses a single, unchanging decryption key can be adaptively secure. Impossibility holds even if secure data erasure is possible.

We show that this limitation can be overcome by *updating the decryption key over time* and making some mild assumptions about the frequency of communication between parties. Using this approach, we construct adaptively-secure, completely non-interactive encryption schemes supporting secure encryption of arbitrarily-many messages from arbitrarily-many senders. Our schemes additionally provide forward security and security against chosen-ciphertext attacks.

Key words: Public-key encryption, adaptive security, forward security, non-committing encryption.

---

*An extended abstract of this work appeared at TCC '05.

†IBM T.J. Watson Research Center, NY, USA.

‡Department of Computer Science, University of Maryland. This work was supported by NSF Trusted Computing Grant #0310751.

Contents

1 Introduction **1**
1.1 Our Contributions . . . 2
1.2 Techniques and Constructions . . . 2
1.3 Organization . . . 3

2 Definition of AFSE **4**
2.1 Handling Multiple Outstanding Ciphertexts . . . 4
2.2 Realizing $\mathcal{F}_{\mathrm{AFSE}}$ Using Key-Evolving Encryption Schemes . . . 6

3 AFSE Based on Forward-Secure Encryption **6**
3.1 Proof of Theorem 1 . . . 8
3.1.1 Conventions and Notation . . . 8
3.1.2 Description of the Simulator . . . 9
3.1.3 Proof of Indistinguishability . . . 11

4 Receiver Non-Committing Encryption **14**
4.1 Definition of RNCE . . . 15
4.2 A Secure RNCE Scheme for Polynomial-Size Message Spaces . . . 16
4.3 A Secure RNCE Scheme for Exponential-Size Message Spaces . . . 18

5 AFSE Based on Receiver Non-Committing Encryption **21**
5.1 Proof of Theorem 7 . . . 22
5.1.1 Description of the Simulator . . . 22
5.1.2 Proof of Indistinguishability . . . 22
5.1.3 Comments . . . 29

A Key-Evolving and Forward-Secure Encryption **31**

B The UC Framework, Abridged **32**
B.1 The Public-Key Encryption Functionality $\mathcal{F}_{\mathrm{PKE}}$ . . . 32

1 Introduction

Imagine a band of political dissidents who need to go into hiding from an oppressive regime. While in hiding, the only form of communication with the outside world is via the public media. Before going into hiding, each individual wants to publish a key that will allow *anyone* (even parties not currently known to this individual) to publish encrypted messages that only this individual can decipher. Since it is not known in advance how long these members will need to be in hiding, reasonably short public keys must suffice for encrypting an unbounded number of messages. Furthermore, messages encrypted to each dissident must remain secret even if other dissidents are caught and their secrets are extracted from them. Do encryption schemes satisfying these requirements exist?

At first glance, a standard public-key encryption scheme seems to suffice. Indeed, a public-key encryption schemes allows a receiver to publish a key that can then be used by anyone to send encrypted messages to the receiver. The public key is *short* (i.e., of fixed polynomial length) and can be used by arbitrary senders (potentially unknown to the receiver at the time the key is published) to securely send arbitrarily-many messages to the receiver *without further interaction*. Furthermore, senders need not maintain any state other than the receiver's public key, and the receiver similarly need not maintain any state except for his secret key.

However, standard public-key encryption schemes do not provide the desired level of security. Standard definitions of security, including semantic security against passive attacks [GM84] as well as various notions of security against active attacks [NY90, RS91, DDN00, BDPR98], only consider the case where the adversary never learns any secret key. However, when an adversary can compromise players and learn their internal states in an *adaptive* manner, possibly depending on previously-observed ciphertexts and information learned during previous corruptions, the standard notions no longer apply. In particular, in the adaptive setting *encrypting with a CCA-secure encryption scheme is not known to provide secure communication.*

To obtain provable security against adaptive adversaries, one must ensure that the information gathered by the adversary when compromising parties (namely, their secret keys) does not give the adversary any additional advantage toward compromising the security of the yet-uncorrupted parties. The standard way of formulating this is by requiring the existence of a *simulator* that can generate "dummy ciphertexts" which can be later "opened" (i.e., by revealing an appropriate secret key) as encryptions of any message; see, e.g., [CFGN96]. A scheme satisfying this additional condition is said to be *adaptively secure.*

Several methods are known for achieving adaptively-secure encrypted communication, but none can be used in the basic setting exemplified by the above toy problem. Beaver and Haber [BH92] propose an adaptively secure encryption protocol in which the sender and receiver must interact before they can securely communicate for the first time. Furthermore, the parties must maintain a shared secret key *per connection.* This key must be continually updated, with the old key being erased, as more messages are encrypted. *Non-committing encryption schemes* [CFGN96, B97, DN00] more closely mimic the functionality of standard public-key encryption, and in particular do not require maintenance of per-connection state. (In addition, these solutions also remove the need for secure data erasure.) In these schemes, however, both the public and secret keys are at least as long as the overall number of bits to be encrypted. In fact, as noted by Nielsen [N02], any adaptively-secure scheme with non-interactive encryption *must* have a decryption key which is at least as long as the number of bits to be decrypted under this key. In a nutshell, this is because the simulator must "open" the "dummy ciphertexts" as encryptions of any given sequence of messages by presenting an appropriate secret key; therefore, the number of possible secret keys must be at least the number of possible message-sequences. The unfortunate conclusion is that a public-key encryption scheme that can encrypt an unbounded number of messages with short and unchanging keys cannot be adaptively secure. This holds even if secure data erasures are possible, and even in a weaker setting where only receivers can be corrupted.

We also comment that previous work on adaptively-secure encryption did not address resistance to chosen-ciphertext attacks.

1.1 Our Contributions

This work demonstrates that we can circumvent Nielsen's negative result if the secret decryption key is allowed to periodically change, and some mild assumptions about the frequency of communication between parties are made. That is, under standard hardness assumptions, there exist adaptively-secure, non-interactive public-key encryption schemes with short keys that can handle arbitrarily-many messages and senders. In particular, our schemes solve the toy example from above in a way that is essentially the best possible under the given constraints.

This is done by considering *key-evolving* encryption schemes [CHK03] in which the secret key is locally updated by the receiver according to a globally-known schedule (say, at the end of every day), while the public key remains fixed. The secret key for the previous period is securely *erased* once it is no longer needed. Using this approach, we construct adaptively-secure, non-interactive encryption schemes that can be used to encrypt arbitrarily-many bits as long as the number of encrypted bits (for any particular key) is bounded *per time period*. As discussed above, an assumption of this sort is essential to circumvent Nielsen's negative results. Also, this assumption is reasonable in many cases: for instance, one may easily posit some known upper bound on the number of incoming e-mails processed per day.

In addition to being adaptively secure, our schemes also provide both forward security [A97, CHK03] and security against chosen-ciphertext attacks. (We comment that although forward security is reminiscent of adaptive security, neither security property implies the other.) Accordingly, we refer to schemes satisfying our security requirements as adaptively- and forward-secure encryption (AFSE) schemes. We formalize the requirements for AFSE schemes within the UC framework [C01]. That is, we present an functionality $\mathcal{F}_{\text{AFSE}}$ that captures the desired properties of AFSE schemes. This functionality is a natural adaptation of the "standard" public-key encryption functionality of [C01, CKN03] to the context of key-evolving encryption. As in the non-adaptive case, $\mathcal{F}_{\text{AFSE}}$ guarantees security against active adversaries, which in particular implies security against chosen-ciphertext attacks. Using the composability properties of the UC framework, our constructions are guaranteed to remain secure in any protocol environment. Indeed, the formulation of $\mathcal{F}_{\text{AFSE}}$, which blends together the notions of forward security, chosen-ciphertext security, and adaptive security of public-key encryption schemes, is another contribution of this work.

1.2 Techniques and Constructions

We first note that dealing with corruption of senders is easy, since a sender can simply erase its local state upon completing the encryption algorithm. We thus concentrate on the more difficult case of receiver corruption. We then show that it suffices to consider AFSE for the case when only a *single* message is encrypted per time period, since any such construction can be extended in a generic manner to give a scheme which can be used to encrypt any bounded number of messages per time period. With this in mind, our first construction uses the paradigm of Naor-Yung and Sahai [NY90, S99] to construct an AFSE scheme based on any forward-secure encryption (FSE) scheme and any simulation-sound non-interactive zero-knowledge (NIZK) proof system [DDOPS01]. Recall that, under the Naor-Yung/Sahai paradigm, the sender encrypts messages by essentially using two independent copies of a semantically-secure encryption scheme together with an NIZK proof of consistency. To decrypt, the receiver verifies the proof and then decrypts either one of the component ciphertexts. Naor and Yung prove that this provides security against "lunch-time" (i.e., non-adaptive) chosen-ciphertext attacks when an arbitrary NIZK proof system is used, and Sahai later showed that this technique achieves full (i.e., adaptive) CCA-security if a one-time simulation-sound NIZK proof system is used. We show that if a semantically-secure FSE scheme is used as the underlying encryption scheme, and the NIZK proof system is "fully" simulation sound (as defined in [DDOPS01]), the resulting construction is also an AFSE scheme. This approach can be extended to encrypt a polynomial number of bits per ciphertext using only a single NIZK proof. (We remark that, as opposed to the case of standard CCA-secure encryption [S99], here it is not enough that the underlying NIZK is *one-time* simulation sound.)

While the above approach is conceptually simple, it is highly impractical due to the inefficiency of known NIZKs. We thus propose an alternate approach that leads to more efficient solutions based on specific, number-theoretic assumptions. As part of this approach, we first define and construct "standard" (i.e., non key-evolving) encryption schemes which are secure against lunch-time chosen-ciphertext attacks and are adaptively-secure for encryption of a *single* message (in total). We call such schemes *receiver non-committing encryption* (RNCE) schemes.[1] Our construction of an AFSE scheme proceeds by first encrypting the message using any RNCE scheme, and then encrypting the resulting ciphertext using any CCA-secure FSE scheme. Informally, this construction achieves adaptive security for an *unbounded* number of messages (as long as only one message is encrypted per time period) because the secret key of the outer FSE scheme is updated after every period and so the simulator only needs to "open" one ciphertext (i.e., the one corresponding to the current time period) as an arbitrary message. It can accomplish the latter using the "inner" RNCE scheme.

Obtaining an efficient scheme using this approach requires efficient instantiation of both components. Relatively efficient CCA-secure FSE schemes (in particular, schemes which avoid the need for NIZK proofs) are already known [CHK04, BB04]. Therefore, we focus on constructing efficient RNCE schemes based on specific number-theoretic assumptions. Our first RNCE scheme is based on the Cramer-Shoup encryption scheme [CS98] (and adapts techniques of [JL00]) and its security is predicated on the decisional Diffie-Hellman (DDH) assumption. However, this scheme allows encryption of only a logarithmic number of bits per ciphertext. We also show a second RNCE scheme based on the schemes of [GL03, CS03] (which, in turn, build on [CS02]), whose security relies on the decisional composite residuosity assumption introduced by Paillier [P99] and which can be used to encrypt a polynomial number of bits per ciphertext.

1.3 Organization

The AFSE functionality is defined and motivated in Section 2. Our construction of AFSE using the Naor-Yung/Sahai paradigm is described in Section 3. In Section 4, we present definitions for RNCE and show two constructions of RNCE schemes based on specific number-theoretic assumptions. Finally, in Section 5 we construct an AFSE scheme from any RNCE scheme and any CCA-secure FSE scheme. In Appendix A, we include definitions of key-evolving and forward-secure encryption, while a brief overview of the UC framework and its application to secure encryption is provided in Appendix B. All proofs of security are also deferred to the appendices.

[1] Indeed, this is a relaxation of the notion of non-committing encryption from [CFGN96]. It is similar to the relaxation studied by Jarecki and Lysyanskaya [JL00], except that we also require security against lunch-time chosen-ciphertext attacks.

2 Definition of AFSE

We define AFSE by specifying an appropriate ideal functionality in the UC security framework (cf. Appendix B). This functionality, denoted $\mathcal{F}_{\mathrm{AFSE}}$ and presented in Figure 1, is obtained by appropriately modifying the "standard" public-key encryption functionality $\mathcal{F}_{\mathrm{PKE}}$ [C01, CKN03] which is reviewed in Appendix B.1.

Intuitively, $\mathcal{F}_{\mathrm{AFSE}}$ captures the same security notions as $\mathcal{F}_{\mathrm{PKE}}$ except that it also provides a mechanism by which the receiver can "update" its secret key; $\mathcal{F}_{\mathrm{AFSE}}$ guarantees security only as long as a bounded number of messages are encrypted between key updates. In fact, for simplicity, the functionality as defined only guarantees security when a *single* ciphertext is encrypted between key updates. Say a ciphertext encrypted with respect to a particular time period t is outstanding until the receiver has updated its secret key a total of $t + 1$ times. Then, if more than one outstanding ciphertext is requested, the functionality guarantees no security whatsoever for this ciphertext. (Formally, this is captured by handing the corresponding plaintext to the adversary.) Section 2.1 discusses how $\mathcal{F}_{\mathrm{AFSE}}$ can be extended to allow any bounded number of outstanding ciphertexts, which corresponds to ensuring security as long as at most this many messages are encrypted between key updates. It also presents a generic transformation from protocols secure for a single outstanding ciphertext to protocols secure for the general case.

For convenience, we highlight some differences between $\mathcal{F}_{\mathrm{AFSE}}$ and $\mathcal{F}_{\mathrm{PKE}}$. First, an additional parameter — a time period t — is introduced. An encryption request now additionally specifies a time period for the encryption called the "sender time", and the functionality maintains a variable t^* called the "receiver time". The receiver time is initialized to 0, and is incremented by the receiver R^* using an Update request. A ciphertext generated for sender time t is only decrypted by $\mathcal{F}_{\mathrm{AFSE}}$ (upon request of the appropriate receiver) when the current receiver time is $t^* = t$.

Second, $\mathcal{F}_{\mathrm{AFSE}}$ limits the information gained by the adversary upon corruption of parties in the system. When corrupting parties other than R^*, the adversary learns nothing. When corrupting R^* at some "receiver time" t^*, the adversary does not learn any information about messages that were encrypted at "sender times" $t < t^*$. (This is akin to the level of security provided by forward-secure encryption schemes, and in fact strengthens the usual notion of adaptive security which potentially allows an adversary to learn all past messages upon corruption of a party.) In addition, *adaptive* security is guaranteed for a single message encrypted at some sender time $t \geq t^*$ (i.e., a single outstanding message).

The fact that security is guaranteed only for a single outstanding message is captured via the variable messages-outstanding, which is initialized to 0 and is set to 1 when a message is encrypted for time period t with $t \geq t^*$. When the receiver's time unit t^* advances beyond the time unit t of the outstanding ciphertext, the variable messages-outstanding is reset to 0. If another encryption request arrives with time period $t \geq t^*$ while messages-outstanding is equal to 1, then $\mathcal{F}_{\mathrm{AFSE}}$ *discloses the entire plaintext to the adversary* (and thus does not ensure any secrecy in this case).

We remark that $\mathcal{F}_{\mathrm{AFSE}}$ can be used in a natural way to realize a variant of the "secure message transmission functionality" [C01, AF04] in synchronous networks with respect to adaptive adversaries. We omit further details.

2.1 Handling Multiple Outstanding Ciphertexts

While the functionality $\mathcal{F}_{\mathrm{AFSE}}$ and all the constructions in this work are described assuming a bound of at most one outstanding ciphertext, both the functionality and the constructions can be generalized to the case of any *bounded* number of outstanding ciphertexts (corresponding to a bounded number of messages encrypted per time period). Generalizing the functionality is straightforward,

Functionality $\mathcal{F}_{\text{AFSE}}$

$\mathcal{F}_{\text{AFSE}}$ proceeds as follows, when parameterized by message domain ensemble $\mathcal{D} = \{D_k\}_{k\in\mathbb{N}}$ and security parameter k.

Key Generation: Upon receiving a request $(\texttt{KeyGen}, sid)$ from party R^*, do: Verify that $sid = (sid', R^*)$ (i.e., that the identity R^* is encoded in the session ID). If not, then ignore this input. If yes:

1. Hand $(\texttt{KeyGen}, sid)$ to the adversary.
2. Receive a value pk^* from the adversary, and hand pk^* to R^*. Initialize $t^* \leftarrow 0$ and $\mathsf{messages\text{-}outstanding} \leftarrow 0$.

Encryption: Upon receiving from some party P a tuple $(\texttt{Encrypt}, sid, \mathsf{pk}, t, m)$ proceed as follows:

1. If $m \in D_k$, $\mathsf{pk} = \mathsf{pk}^*$, and either $t < t^*$ or $\mathsf{messages\text{-}outstanding} = 0$, then send $(\texttt{Encrypt}, sid, \mathsf{pk}, t, P)$ to the adversary. In all other cases, send $(\texttt{Dummy-Encrypt}, sid, \mathsf{pk}, t, m, P)$ to the adversary (i.e., reveal the plaintext to the adversary).
2. Receive a reply c from the adversary and send $(\texttt{ciphertext}, c)$ to P. In addition, if $m \in D_k$, $\mathsf{pk} = \mathsf{pk}^*$, and $t \geq t^*$, then do:
   (a) If $\mathsf{messages\text{-}outstanding} = 0$, set $\mathsf{messages\text{-}outstanding} \leftarrow 1$ and $\mathsf{flag} \leftarrow \mathsf{outstanding}$. Else, set $\mathsf{flag} \leftarrow \mathsf{dummy}$.
   (b) record the tuple (m, t, c, flag) in the list of ciphertexts.

Decryption: Upon receiving a tuple $(\texttt{Decrypt}, sid, c)$ from player P, if $P \neq R^*$ then ignore this input. Otherwise:

1. If the list of ciphertexts contains a tuple $(m, t, c, \star)$ with the given ciphertext c and $t = t^*$, then return m to R^*.
2. Otherwise send a message $(\texttt{Decrypt}, sid, t^*, c)$ to the adversary, receive a reply m, and forward m to R^*.

Update: Upon receiving $(\texttt{Update}, sid)$ from player P, if $P = R^*$ do:

1. Send a message $(\texttt{Update}, sid)$ to the adversary.
2. Remove from the list of ciphertexts all the tuples $(m, t^*, c, \mathsf{flag})$ with the current time t^*. If any of these tuple has $\mathsf{flag} = \mathsf{outstanding}$, then reset $\mathsf{messages\text{-}outstanding} \leftarrow 0$.
3. Set $t^* \leftarrow t^* + 1$.

Corruptions: Upon corruption of party P, if $P = R^*$ then send to the adversary all tuples $(m, t, c, \star)$ in the list of ciphertexts with $t \geq t^*$. (If $P \neq R^*$ then do nothing.)

Figure 1: The AFSE functionality, $\mathcal{F}_{\text{AFSE}}$.

so we do not describe it here. As for constructions, any AFSE scheme which is secure for the case of a single outstanding ciphertext can be extended generically so as to be secure for any bounded number ℓ of outstanding ciphertext in the following way: The public key of the new scheme consists of ℓ independent keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell$ generated using the original scheme. To encrypt a message m, the sender computes the "nested encryption" $E_{\mathsf{pk}_1}(E_{\mathsf{pk}_2}(\cdots E_{\mathsf{pk}_\ell}(m)\cdots))$ and sends the resulting ciphertext to the receiver. One can show that this indeed realizes $\mathcal{F}_{\mathrm{AFSE}}$ for at most ℓ outstanding ciphertexts. The formal proof, however, is more involved and is omitted.

2.2 Realizing $\mathcal{F}_{\mathrm{AFSE}}$ Using Key-Evolving Encryption Schemes

We present our constructions as key-evolving encryption *schemes* (i.e., as a collection of algorithms) rather than as protocols (as technically required by the UC framework). For completeness, we describe the (obvious) transformation from key-evolving encryption schemes to protocols geared toward realizing $\mathcal{F}_{\mathrm{AFSE}}$.

Recall that a key-evolving encryption scheme consists of four algorithms $(\mathsf{Gen}, \mathsf{Upd}, \mathsf{Enc}, \mathsf{Dec})$, where $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ are the key generation, encryption, and decryption routines (as in a standard encryption scheme, except that the encryption and decryption routines also take as input a time period t), and Upd is the secret-key update algorithm that takes as input the current secret key and time unit, and outputs the secret key for the next time unit. The definition is reviewed in Appendix A.

Given a key evolving encryption scheme $S = (\mathsf{Gen}, \mathsf{Upd}, \mathsf{Enc}, \mathsf{Dec})$, one may construct the protocol π_S as follows:

1. When activated within player R^* and with input $(\texttt{KeyGen}, sid, 1^k)$, run algorithm $\mathsf{Gen}(1^k)$, output the encryption key pk, record the decryption key sk_0, initialize a counter $t \leftarrow 0$, and erase all other local state.

2. When activated within R^* with input $(\texttt{Update}, sid)$, set $\mathsf{sk}_{t+1} \leftarrow \mathsf{Upd}(t, \mathsf{sk}_i)$, set $t \leftarrow t+1$, and erase sk_i (as well as any other temporary state that was used for the Upd algorithm).

3. When activated within some player P with input $(\texttt{Encrypt}, sid, \mathsf{pk}', t', m)$, choose a random r and return $\mathsf{Enc}_{\mathsf{pk}'}(t', m, r)$, and then erase all local state. (Note that it does not necessarily hold that $\mathsf{pk}' = \mathsf{pk}$.)

4. When activated within R^* with input $(\texttt{Decrypt}, sid, c)$, return $\mathsf{Dec}_{\mathsf{sk}}(t, c)$ (and then erase any temporary state that was used for the decryption).

With this transformation we can now define an AFSE scheme:

Definition 1 *A key-evolving encryption scheme S is an adaptively- and forward-secure encryption (AFSE) scheme if the protocol π_S resulting from the transformation above securely realizes $\mathcal{F}_{\mathrm{AFSE}}$ with respect to adaptive adversaries.*

3 AFSE Based on Forward-Secure Encryption

In this section we show how to construct an AFSE scheme from any FSE scheme secure against chosen-plaintext attacks along with any simulation-sound NIZK proof system. (See Appendix A for definitions of key-evolving encryption and forward security, both against chosen-plaintext and chosen-ciphertext attacks.) We describe in detail a construction that allows encryption of only a single bit per ciphertext and then discuss how this may be generalized to allow for encryption of any polynomial number of bits per ciphertext. Our construction uses a simple twist of the Naor-Yung/Sahai transformation [NY90, S99]; when applied to two FSE schemes, the resulting scheme yields not only CCA security but also security against adaptive corruptions. We comment that, as opposed to the case of non-adaptive CCA security, "one-time" simulation sound NIZK proofs are *not sufficient* to achieve security against adaptive corruptions; instead, we require NIZK proofs satisfying the stronger notion of unbounded simulation soundness [DDOPS01].

The construction. Let $\mathcal{E}' = (G', U', E', D')$ be a key-evolving encryption scheme, and let $\mathcal{P} = (\ell, P, V)$ be an NIZK proof system (where $\ell(k)$ is the length of the common random string for security parameter k) for the following NP language

$$\begin{aligned} L_{\mathcal{E}'} \stackrel{\text{def}}{=} \{&(t, \mathsf{pk}'_0, c'_0, \mathsf{pk}'_1, c'_1) \ : \\ &\exists\, m, r_0, r_1 \text{ s.t. } c'_0 = E'(\mathsf{pk}'_0, t; m; r_0),\ c'_1 = E'(\mathsf{pk}'_1, t; m; r_1)\}. \end{aligned}$$

We construct a new key-evolving encryption scheme $\mathcal{E} = (G, U, E, D)$ as follows:

Key generation, G. On security parameter 1^k, run two independent copies of the key generation algorithm of $\mathcal{E}'$ to obtain $(\mathsf{pk}'_0, \mathsf{sk}'_0) \leftarrow G'(1^k)$ and $(\mathsf{pk}'_1, \mathsf{sk}'_1) \leftarrow G'(1^k)$. Choose a random bit $b \in \{0,1\}$ and a random $\ell(k)$-bit string $\mathsf{crs} \in \{0,1\}^{\ell(k)}$. The public key is the triple $(\mathsf{pk}'_0, \mathsf{pk}'_1, \mathsf{crs})$, and the secret key is (b, sk'_b). Erase the other key $\mathsf{sk}'_{\bar{b}}$.

Key update, U. Key update is unchanged, namely $U(t, (b, \mathsf{sk}')) = (b, U'(t, \mathsf{sk}'))$.

Encryption, E. To encrypt a bit $m \in \{0,1\}$ at time t, first pick two independent random strings r_0, r_1 as needed for the encryption algorithm E' and compute $c'_0 \leftarrow E'(\mathsf{pk}'_0, t; m; r_0)$, $c'_1 \leftarrow E'(\mathsf{pk}_1, t; m; r_1)$, and a proof that $(t, \mathsf{pk}'_0, c'_0, \mathsf{pk}'_1, c'_1) \in L_{\mathcal{E}'}$ (namely, compute $\pi \leftarrow P(\mathsf{crs}; t, \mathsf{pk}'_0, c'_0, \mathsf{pk}'_1, c'_1; m, r_0, r_1)$). The ciphertext is the triple $c = (c'_0, c'_1, \pi)$.

Decryption, D. To decrypt a ciphertext $c = (c'_0, c'_1, \pi)$ at time t, first run the verifier $V(\mathsf{crs}; t, \mathsf{pk}'_0, c'_0, \mathsf{pk}'_1, c'_1)$. If V rejects, the output is $\perp$. Otherwise, the recipient uses (b, sk'_b) to recover $m \leftarrow D'(\mathsf{sk}'_b; c'_b)$.

We claim the following theorem:

Theorem 1 *If $\mathcal{E}'$ is forward-secure against chosen-plaintext attacks (*fs-CPA*, cf. Definition 4) and if (P, V) is an unbounded simulation-sound NIZK proof system* [DDOPS01, Def. 6], *then $\mathcal{E}$ is an AFSE scheme.*

Before proceeding with the proof, we provide some intuition. Underlying our analysis is the observation that a simulator (who can generate proofs for false assertions) can come up with a valid-looking "dummy ciphertext" whose component ciphertexts encrypt *different* messages (i.e., both 0 and 1). The simulator, who also knows both underlying decryption keys, can thus open the dummy ciphertext as an encryption of either 0 or 1, depending on which decryption key is presented to an adversary. (Note further that the adversary will be unable to generate dummy ciphertexts of this form due to the simulation soundness of the NIZK proof system.) The above argument demonstrates adaptive security for a single encrypted bit. Adaptive security for an unbounded number of bits (as long as only one ciphertext is outstanding) holds since the secret keys of the underlying FSE schemes evolve after each encryption. We remark that one-time simulation soundness for (P, V)

would not be sufficient here, since the simulator must generate *multiple* "fake ciphertexts" and the hybrid argument that works in the non-adaptive case (see [S99]) does not work here.

AFSE for longer messages. To obtain a construction of an AFSE scheme for n-bit messages, one can simply use n pairs of public keys generated using $\mathcal{E}'$ (the receiver now chooses at random one secret key from each pair to store, while the other is erased). The rest is an obvious extension of the proof intuition from above, with the only subtle point being that the resulting ciphertext contains a single NIZK proof computed over the entire vector of n ciphertext pairs (with the language being defined appropriately).

3.1 Proof of Theorem 1

To prove this theorem, we describe a simulator and prove that no PPT environment can distinguish interactions with the "standard real-world adversary" and the scheme $\mathcal{E}$ from interactions with the ideal-world simulator and the functionality $\mathcal{F}_{\text{AFSE}}$ (with domain $\mathcal{D} = \{0,1\}$). Before doing so, we establish some useful conventions and notation that will be used here as well as in later proofs.

3.1.1 Conventions and Notation

Recall that in this work we are dealing with a *non-interactive scheme*, where parties never communicate directly with each other (any interaction in a run of the system is done via the environment). This means that direct interaction between the environment and the adversary (in either model) happens only when players are corrupted. Given the functionality $\mathcal{F}_{\text{AFSE}}$, the relevant interface between the environment, the adversary, and the system, consists of the following queries:

- The environment sends a query $(\texttt{KeyGen}, sid)$ to player i, which is answered by some public key pk. Below we call the player whose identity is encoded in the sid "the receiver", denoted R^*. The public key given to the receiver is called the "target public key" and is denoted by pk^*. We refer to t^* (which is initialized by $\mathcal{F}_{\text{AFSE}}$ in response to this query and later incremented with every update query) as the "local time at the receiver".

- The environment sends a query $(\texttt{Update}, sid)$ to the receiver R^*, but this query generates no response.

- The environment sends a query $(\texttt{Encrypt}, sid, \mathsf{pk}, t, m)$ to some player j, which is answered by a ciphertext c. We call the time t included in this query the "sender time" for the query. We say an encryption query is $\mathsf{important}$ if it causes the $\mathcal{F}_{\text{AFSE}}$ functionality to send a message $(\texttt{Encrypt}, sid, \mathsf{pk}, t, j)$ to the simulator. (That is, m is kept hidden from the adversary.) That means in particular that: (a) the query specifies the target public key pk^*, (b) the message m is in the domain D, (c) the receiver is not corrupted, and (d) either the sender time is "in the past" (i.e., $t < t^*$), or the variable $\mathsf{messages\text{-}outstanding}$ is not set when this query is made. We comment that from the transcript of the environment, it is easy to deduce which encryption queries are important.

- The environment sends a query $(\texttt{Decrypt}, sid, c)$ to the receiver R^*, which is answered by a plaintext m (possibly $m = \perp$).

  A decryption query is $\mathsf{important}$ if it causes the $\mathcal{F}_{\text{AFSE}}$ functionality to send a message $(\texttt{Decrypt}, sid, t^*, c, i)$ to the simulator. This roughly means that the ciphertext c was *not* generated in response to a previous encryption query. Again, from the transcript of the environment, it is easy to deduce which decryption queries are important.

- The environment sends a query $(\texttt{Corrupt}, sid, i)$ to the adversary. In the real world, the adversary is then supposed to corrupt player i and send its internal state to the environment. (Presumably, the relevant part of the internal state consists of a key, say of the form $(\mathsf{pk}, t, \mathsf{sk})$.)

Of course, the environment is free to issue other queries to the players, but these will simply be ignored. Also, in the real world it is sufficient to consider a "dummy adversary" that only relays messages from the environment to $\mathcal{F}_{\text{AFSE}}$ and back (see [C01]).

Additional notation and observations. A ciphertext c which is returned as a reply to an important encryption query (pk^*, t, m) is said to be "outstanding" as long as the local time at the receiver is $t^* \leq t$. We call a ciphertext that is outstanding when the receiver is corrupted "the critical ciphertext", and the encryption query that resulted in that ciphertext is called "the critical encryption query". Note that the $\mathcal{F}_{\text{AFSE}}$ functionality ensures that at any time there is at most one outstanding ciphertext in the system, and in particular there can be at most one critical ciphertext. In fact, we can assume that in every run there is a critical ciphertext, since we can require w.l.o.g. that the environment always corrupts the receiver and, just before doing so, makes an encryption query with the current local time t^*.

We also observe that if a ciphertext c is outstanding at local time t^*, then every important encryption query that was issued up to local time t^* (other than the one that generated c) must have sender time $t' < t^*$. (Otherwise, we would have two outstanding ciphertexts at local time t^*.) In particular, if the receiver is corrupted at local time t^* then all the important encryption queries that are made before the receiver was corrupted — except the critical query — must have sender time $t' < t^*$.

3.1.2 Description of the Simulator

The simulator is quite natural. Roughly, to generate a key it uses the NIZK simulator to generate the common random string crs, and it runs the original key-generation G' twice and keeps both secret keys sk'_0 and sk'_1. When it needs to generate a ciphertext (and it does not see the bit to be encrypted), it picks a bit b at random, encrypts b under pk'_0 and $\bar{b}$ under pk'_1, and uses the NIZK simulator to generate a simulated proof for these two ciphertexts. If the simulator needs to decrypt ciphertexts that were generated by the environment, it verifies the NIZK proof and (if correct) decrypts the ciphertext using either of the secret keys that it knows. Finally, if the receiver is compromised and the simulator needs to open one of the invalid ciphertexts that it created, it learns the bit that was supposed to be encrypted in that ciphertext and then reveals either sk'_0 or sk'_1 (depending on which decrypts the ciphertext to the right bit). Note that the simulator never needs to open more than one of these invalid ciphertexts, since there can be at most one critical ciphertext. A detailed description follows.

Recall that we are dealing with a *non-interactive scheme*, where parties never communicate directly with each other (any interaction in a run of the system is done via the environment). This means that direct interaction between the environment and the adversary (in either model) happens only when players are corrupted. Thus, when specifying the simulator we only need to show how it handles corrupted players and how it interacts with the functionality $\mathcal{F}_{\text{AFSE}}$. Below we list all the types of interactions that the simulator needs to handle, and how it implements them. Let $S = (S_1, S_2)$ be the NIZK simulator that is guaranteed to exist in [DDOPS01, Def. 6].

Key Generation. Upon receiving a message $(\texttt{KeyGen}, sid, i, 1^k)$ from the $\mathcal{F}_{\text{AFSE}}$ functionality, if this is the first activation then do the following:

- Run two independent copies of the key generation of $\mathcal{E}'$ to get $(\mathsf{pk}'_0, \mathsf{sk}'_0) \leftarrow G'(1^k)$ and $(\mathsf{pk}'_1, \mathsf{sk}'_1) \leftarrow G'(1^k)$.
- Run the first stage of the NIZK simulator to get $(\mathsf{crs}, z) \leftarrow S_1(1^k)$.
- Return to the functionality the public key $\mathsf{pk} = (\mathsf{pk}'_0, \mathsf{pk}'_1, \mathsf{crs})$.
- Record the variables $\mathsf{pk}^* \leftarrow \mathsf{pk}$, $R^* \leftarrow i$, $t^* \leftarrow 0$, $\mathsf{sk}^*_0 \leftarrow \mathsf{sk}'_0$, $\mathsf{sk}^*_1 \leftarrow \mathsf{sk}'_1$ and $z^* \leftarrow z$.

If this is not the first activation, just run the key-generation of $\mathcal{E}$ to get $(\mathsf{pk}, \mathsf{sk}) \leftarrow G(1^k)$ and return pk to the functionality.

Update. Upon receiving a message $(\texttt{Update}, sid)$ from the functionality, update the secret keys $\mathsf{sk}^*_0, \mathsf{sk}^*_1$ using the update algorithm of $\mathcal{E}'$, setting $\mathsf{sk}^*_0 \leftarrow U'(t^*, \mathsf{sk}^*_0)$ and $\mathsf{sk}^*_1 \leftarrow U'(t^*, \mathsf{sk}^*_1)$, and then set $t^* \leftarrow t^* + 1$.

Encryption. Upon receiving a message $(\texttt{Encrypt}, sid, \mathsf{pk}, t, j)$ from the functionality (with $\mathsf{pk} = \mathsf{pk}^*$), parse pk as $(\mathsf{pk}'_0, \mathsf{pk}'_1, \mathsf{crs})$ and do the following:

- Pick at random a bit b, set $c'_0 \leftarrow E'(\mathsf{pk}'_0, t; b)$ and $c'_1 \leftarrow E'(\mathsf{pk}'_1, t; \bar{b})$.
- Compute a simulated proof $\pi \leftarrow S_2(z^*, t, \mathsf{pk}'_0, c'_0, \mathsf{pk}'_1, c'_1)$.
- Return the ciphertext $c = (c'_0, c'_1, \pi)$ to the functionality, and record this ciphertext together with the bit b.

Upon receiving a message $(\texttt{Dummy-Encrypt}, sid, \mathsf{pk}, t, m, j)$ from the functionality, simply apply the encryption routine of $\mathcal{E}$ to get $c \leftarrow E(\mathsf{pk}, t; m)$ and return c.

Decryption. Upon receiving a message $(\texttt{Decrypt}, sid, t^*, c, i)$ from the functionality, if $i \neq R^*$ then return $\perp$. Otherwise, parse $\mathsf{pk}^* = (\mathsf{pk}'_0, \mathsf{pk}'_1, \mathsf{crs})$ and $c = (c'_0, c'_1, \pi)$, and do the following:

- Verify the proof π using $V(\mathsf{crs}, t^*, \mathsf{pk}'_0, c'_0, \mathsf{pk}'_1, c'_1, \pi)$.
- If verification succeeds then decrypt $m \leftarrow D'(\mathsf{sk}^*_0; c'_0)$ and if $m \in \{0,1\}$ then return m.
- If verification fails or $m \notin \{0,1\}$ then return $\perp$.

Corruptions. When the simulator is instructed by the environment to corrupt the receiver R^*, it informs the functionality that R^* is corrupted, and gets from it all the tuples (m, t, c, flag) with $t \geq t^*$. (Recall that there can be at most one such tuple which is outstanding. Also, since the message domain is $D = \{0,1\}$, then m must be a single bit.) Then it does the following:

- If there is no outstanding tuple, pick a random bit σ and set $\mathsf{sk}^* \leftarrow (\sigma, \mathsf{sk}^*_\sigma)$.
- If there an outstanding tuple $(m, t, c, \mathsf{outstanding})$, then recover the bit b that was recorded with the ciphertext c. If $m = b$ then set $\mathsf{sk}^* \leftarrow (0, \mathsf{sk}^*_0)$. Otherwise, set $\mathsf{sk}^* \leftarrow (1, \mathsf{sk}^*_1)$.
- Send $(\mathsf{pk}^*, t^*, \mathsf{sk}^*)$ to the environment, and then erase all the variables R^*, pk^*, t^*, sk^*_0, sk^*_1, and z^*.

3.1.3 Proof of Indistinguishability

We now prove that the interaction of the environment with the scheme $\mathcal{E}$ and the "standard real-world adversary" is indistinguishable from its interaction with the simulator and the functionality $\mathcal{F}_{\text{AFSE}}$. Below, we assume w.l.o.g. that the environment halts immediately after corrupting the receiver (since at this point, neither the functionality nor the simulator has any secrets that the environment does not know). There are four differences between the real and the simulated worlds:

- In the real world, the public key contains a CRS that was chosen according to the specified distribution, whereas in the simulated world the CRS was generated by the NIZK simulator.
- In the real world, ciphertexts that are encrypted under pk^* include two encryptions of the same bit, whereas in the simulated world they include encryptions of two different bits.
- In the real world, these ciphertexts include a "real proof" (generated by the prover with respect to a "real common random string") that the two component ciphertexts encrypt the same bit. In the simulated world, ciphertexts instead include a simulated proof (generated by the simulator with respect to the "simulated common random string" in the public key).
- In the real world, decryption queries with a valid NIZK proof are decrypted with the same key that is later revealed when R^* is corrupted. In the simulated world, they are decrypted with the key sk_0^*.

To deal with the first two differences we introduce a "hybrid world", in which ciphertexts that are encrypted under pk^* include two encryptions of the same bit, but with a simulated proof (with respect to a "simulated common random string") rather than a real proof. We then show that: (a) the simulated world cannot be distinguished from the hybrid world because of the CPA-security of the underlying scheme $\mathcal{E}'$ and the simulation-soundness of the NIZK proof system (Proposition 2 and Claim 3), and (b) the hybrid world cannot be distinguished from the real world because the proof system is zero-knowledge (Proposition 4). We note that proving indistinguishability between the hybrid and real world is straightforward, while the other indistinguishability proof is somewhat more subtle (as it has to deal with the forward-security aspect of the scheme).

The hybrid world. The hybrid world proceeds like the simulated world (and, in particular, the crs^* component of the target public key is generated by the NIZK simulator), except that important encryption queries are handled differently. Namely, a message $(\texttt{Encrypt}, sid, \mathsf{pk}, t, j)$ that the functionality sends to the simulator (in response to encryption query (pk, t, m) to player i with $m \in \{0,1\}$) is answered by setting $c_0 \leftarrow E(\mathsf{pk}_0^*, t; m)$ and $c_1 \leftarrow E(\mathsf{pk}_1^*, t; m)$, then computing $\pi \leftarrow S_2(z^*, t, \mathsf{pk}_0^*, c_0, \mathsf{pk}_1^*, c_1)$ and returning $c = (c_0, c_1, \pi)$. When the environment corrupts the receiver, the simulator picks at random one of the two secret keys it knows and gives it to the environment.

Proposition 2 (simulated ≈ hybrid) *For any* PPT *environment* Env*, the following is negligible:*

$$\alpha(k) \stackrel{\text{def}}{=} \left| \Pr_{\text{sim}}[\mathsf{Env}(1^k) \to 1] \; - \; \Pr_{\text{hybrid}}[\mathsf{Env}(1^k) \to 1] \right|.$$

Proof. Assume that there is a PPT environment Env for which α is not negligible, and we show a contradiction to the CPA security of the underlying encryption scheme $\mathcal{E}'$. Specifically, we describe an adversary A against the scheme $\mathcal{E}'$ that works as follows: it receives a public key pk', generated by G', it specifies a time t and receives a ciphertext $c' \leftarrow E'_{\mathsf{pk}'}(t, m)$ for a random bit m, together with the secret key sk' for time $t+1$ (i.e., the result of making $t+1$ updates to the original secret key). The goal of A is to guess the bit m, and we show that the advantage of A in guessing m is polynomially related to α.

Let $Q(k)$ be a polynomial that bounds the number of important encryption queries that Env makes on security parameter 1^k. Upon receipt of the public key pk', A begins by picking a random index $q^* \in [1 \ldots Q(k)]$. Then A runs the environment $\mathsf{Env}(1^k)$, creating for it a world that looks like the hybrid world up to encryption query $q^* - 1$, like the simulated world from query $q^* + 1$ and on, and where what happens in query q^* depends on the bit that is encrypted in c'. In more details, A implements both the functionality $\mathcal{F}_{\mathrm{AFSE}}$ and the ideal-world simulator as described above, with the following exceptions.

- When Env makes its first "KeyGen" query to player R^*, A picks a bit σ, sets $\mathsf{pk}^*_\sigma \leftarrow \mathsf{pk}'$, $\mathsf{sk}^*_\sigma \leftarrow$ "?", and $(\mathsf{pk}^*_{\bar\sigma}, \mathsf{sk}^*_{\bar\sigma}) \leftarrow G'(1^k)$. It also computes $(\mathsf{crs}^*, z^*) \leftarrow S_1(1^k)$ and returns to Env the public key $\mathsf{pk}^* = (\mathsf{pk}^*_0, \mathsf{pk}^*_1, \mathsf{crs}^*)$.

- When Env makes its j'th important encryption query (pk, t, m) to player i, causing the functionality to send a message $(\mathtt{Encrypt}, sid, \mathsf{pk}, t, j)$ to the simulator, A responds as follows: If $j < q^*$ then A responds just like in the hybrid world, encrypting twice the bit m and providing a simulated NIZK proof. If $j > q^*$ then A responds just like in the simulated world, encrypting two different bits (in random order) and providing a simulated NIZK proof.

  If $j = q^*$ then A makes its challenge query, specifying the time t and getting back a ciphertext c' and the secret key sk', corresponding to public key pk^*_σ and time $t+1$. If $t < t^*$ (where t^* is the local time of the verifier; i.e., this encryption query is for some time period "in the past") then A computes sk^*_σ by applying the update algorithm U' sufficiently many times to the secret key sk' that it got. Otherwise ($t \geq t^*$), A just records the secret key sk' and the corresponding time $t' \leftarrow t+1$. In either case, it sets $c_\sigma \leftarrow c'$, $c_{\bar\sigma} \leftarrow E'(\mathsf{pk}^*_{\bar\sigma}, t; m)$, and $\pi \leftarrow S_2(z^*, t, \mathsf{pk}^*_0, c_0, \mathsf{pk}^*_1, c_1)$, returns to the functionality (and Env) the ciphertext $c = (c_0, c_1, \pi)$, and record the encrypted bit $m^* \leftarrow m$.

- When Env makes an $\mathtt{Update}$ query, A updates the secret keys that it knows, setting $\mathsf{sk}^*_{\bar\sigma} \leftarrow U'(t^*, \mathsf{sk}^*_{\bar\sigma})$, and if $\mathsf{sk}^*_\sigma \neq$ "?" then it also sets $\mathsf{sk}^*_\sigma \leftarrow U'(t^*, \mathsf{sk}^*_\sigma)$. If A doesn't yet know sk^*_σ but it has recorded sk' and $t' = t^* + 1$, then A sets $\mathsf{sk}^*_\sigma \leftarrow \mathsf{sk}'$. In either case, the local time at the receiver is then updated $t^* \leftarrow t^* + 1$.

- When Env makes an important decryption query $(\mathtt{Decrypt}, sid, c)$ with $c = (c_0, c_1, \pi)$, causing the functionality to send a message $(\mathtt{Decrypt}, sid, t^*, c, R^*)$ to the simulator, then A verifies the NIZK proof π, decrypts $c_{\bar\sigma}$ to get $m \leftarrow D'(\mathsf{sk}^*_{\bar\sigma}; c_{\bar\sigma})$ and verifies that m is a single bit. If everything succeeds it returns m, and otherwise it returns $\perp$.

- When Env makes a query ($\mathtt{Corrupt}, sid, R^*$) to the simulator, if A already knows both sk_σ and $\mathsf{sk}_{\bar\sigma}$, then it just proceeds like the simulator, returning the key that correctly decrypts the (single) outstanding ciphertext (or a random key, if they both decrypt correctly). If A still does not know sk_σ then it returns the secret key $\mathsf{sk}^* \leftarrow (\bar\sigma, \mathsf{sk}_{\bar\sigma})$. (Below we show that in the latter case, $\mathsf{sk}_{\bar\sigma}$ must indeed decrypt correctly the critical ciphertext.)

If Env halts with output 1, then A guesses the bit m^* (i.e., the bit that Env asked to encrypt when A made its challenge query). If Env outputs anything else, then A guesses the complement of m^*.

We start the analysis of A by examining decryption queries. Denote by $\mathsf{BadProof}$ the event in the execution of A where Env makes an important decryption query $(\mathtt{Decrypt}, sid, c)$ with $c = (c_0, c_1, \pi)$, such that the NIZK verifier falsely accepts the proof π. Namely, if the local time at the receiver when this query is made was t^*, then we have $V(\mathsf{crs}^*, t^*, \mathsf{pk}_0^*, c_0, \mathsf{pk}_1^*, c_1, \pi) = \mathsf{accept}$, but $(t^*, \mathsf{pk}_0^*, c_0, \mathsf{pk}_1^*, c_1) \notin L_{\mathcal{E}'}$. We notice that as long as the event $\mathsf{BadProof}$ does not happen, the replies that Env get for its decryption queries are independent of which of the two keys is used to decrypt the ciphertext. Hence, what A returns is the same as the replies that Env gets in the simulated world (which is the same as in the hybrid world). The following claim, which is proven later, follows easily from the unbounded simulation-soundness of the NIZK proof system.

Claim 3 $\Pr_{\mathsf{A}}[\mathsf{BadProof}]$ *is negligible in the security parameter k, where the probability is taken over all the randomness in the execution of* A *(including the choice of its input public key).*

Next we show that if the receiver is corrupted at a time when A only knows one secret key, then this key must indeed decrypt correctly the critical ciphertext. (That is, the key sk^* that A returns decrypts the critical ciphertext to the bit that Env specified in the critical encryption query.) Specifically, we show that in this case the critical encryption query was query $j \leq q^*$ (which means that A explicitly encrypted under $\mathsf{pk}_{\bar{\sigma}}^*$ the bit that Env specified in that query). Assume to the contrary that the critical encryption query was $j > q^*$. Let t be the sender time that was specified in encryption query q^* (which was made before query j) and let t^* denote the local time of the receiver when it was corrupted. Since A knows only one secret key at time t^*, it must be that $t^* \leq t$. But this means that the ciphertext from query q^* is also outstanding at time t^*, contradicting the fact that there could be at most one outstanding ciphertext at any given time. We conclude that conditioned on the event $\mathsf{BadProof}$ not happening:

- The view of Env when $q^* = 1$ and the bit encrypted in the ciphertext c' is m^*, is distributed identically to its view in the simulated world.

  Namely, important encryption queries are answered by encryptions of two different bits (in random and independent order) with a simulated proof, important decryption queries are answered as in the simulated world (which is the same as in the hybrid world), and a corruption is answered with the key that decrypts the outstanding ciphertext to the right bit.

- The view of Env when $q^* = Q(k)$ and the bit encrypted in the ciphertext c' is the complement of m^*, is distributed identically to its view in the hybrid world.

  Namely, important encryption queries are answered by two encryptions of the right bit with a simulated proof, important decryption queries are answered as in the simulated world (which is the same as in the hybrid world), and a corruption is answered with one of the two keys, randomly chosen between them (since σ is chosen at random).

- For any $j \in [2 \dots Q(k)]$, the view of Env when $q^* = j - 1$ and the bit encrypted in the ciphertext c' is m^*, is distributed identically to its view when $q^* = j$ and the bit encrypted in the ciphertext c' is the complement of m^*.

  Namely, important encryption queries up to query j are answered by two encryptions of the right bit and the other important encryption queries (if there are any) are answered with encryptions of two different bits in random and independent order, all with simulated proofs. Important decryption queries are answered as in the simulated world (which is the same as in the hybrid world). Also, if a corruption happens while the outstanding ciphertext consists of two encryptions of the same bit, it is answered with either one of the two keys, randomly chosen between them, while if a corruption happens when the outstanding ciphertext consists of encryptions of two different bits then it is answered with the key that decrypts the outstanding ciphertext to the right bit.

We conclude that the advantage of A in guessing the bit encrypted in c' is at least $\alpha(k)/Q(k) - \Pr_{\mathsf{A}}[\mathsf{BadProof}]$. ■

Proof of Claim 3. The proof is straightforward. We show a forger F that interacts with the NIZK simulator $S = (S_1, S_2)$ and produces a proof of an incorrect theorem, different from all theorems it received from S, with probability exactly $\Pr_{\mathsf{A}}[\mathsf{BadProof}]$. The forger F gets as input a "simulated common random string" crs, generated by S_1. It sets $(\mathsf{pk}', \mathsf{sk}') \leftarrow G'(1^k)$ and then runs A on the input pk', putting the input crs in the target public key. Whenever A needs to generate a simulated proof, F uses its access to S_2 to obtain that proof. Whenever the environment in the execution of A makes an important decryption query (pk, c) with $\mathsf{pk} = (\mathsf{crs}, \mathsf{pk}'_0, \mathsf{pk}'_1)$ and $c = (c'_0, c'_1, \pi)$ at time t^*, the forger F checks if the event $\mathsf{BadProof}$ happened. If so, then F outputs the "false theorem" $(t^*, \mathsf{pk}'_0, c'_0, \mathsf{pk}'_1, c'_1)$ with the proof π. We note that F can indeed check whether $\mathsf{BadProof}$ occurs since it knows the two secret keys.[2] By definition of event $\mathsf{BadProof}$, the false theorem is different from all previous theorems that F received from S_2, and the proof π is accepted (with respect to the input crs). ■

Proposition 4 (hybrid ≈ real) *For any* PPT *environment* Env*, the following is negligible:*

$$\beta(k) \stackrel{\text{def}}{=} \left| \Pr_{\text{hybrid}}[\mathsf{Env}(1^k) \to 1] - \Pr_{\text{real}}[\mathsf{Env}(1^k) \to 1] \right|.$$

Proof. This follows by definition from the (unbounded) zero-knowledge of the proof system. We just note that the combination of the environment, the functionality $\mathcal{F}_{\text{AFSE}}$, and the hybrid-world simulator constitute a distinguisher between the NIZK simulator and the prescribed prover. This is because in either world, the answers to the important encryption queries of the environment consist of *true theorems*, and the only difference between the worlds is that in the hybrid world the proofs for these true theorems are generated by the simulator, while in the real world they are generated by the prescribed prover. ■

4 Receiver Non-Committing Encryption

This section defines and constructs receiver non-committing encryption (RNCE) that is secure against "lunch-time attacks" (aka CCA1-secure). We note that RNCE was considered in [JL00] for the more basic case of chosen-plaintext attacks. Section 5 shows how to combine any RNCE scheme with any FSE scheme secure against chosen-ciphertext attacks to obtain a secure AFSE scheme. Since our proposed constructions of RNCE schemes are quite efficient (and since relatively-efficient constructions of FSE schemes secure against chosen-ciphertext attacks are known [CHK03, CHK04, BB04]), we obtain (relatively) efficient AFSE schemes.

On a high level, a receiver non-committing encryption scheme is one in which a simulator can generate a single "fake ciphertext" and later "open" this ciphertext (by showing an appropriate secret key) as any given message. These "fake ciphertexts" should be indistinguishable from real ciphertexts, even when an adversary is given access to a decryption oracle before the fake ciphertext is known.

[2]Here, we assume that the encryption scheme $\mathcal{E}'$ has no decryption errors.

4.1 Definition of RNCE

Formally, a receiver non-committing encryption (RNCE) scheme consists of five PPT algorithms $(G, E, D, \tilde{F}, \tilde{R})$ such that:

- The randomized key-generation algorithm G takes as input the security parameter and outputs a key-pair and some auxiliary information. This is denoted by: $(\mathsf{pk}, \mathsf{sk}, z) \leftarrow G(1^k)$. The public key pk defines a message space $\mathcal{D}$.
- The randomized encryption algorithm E takes a public key pk and a message $m \in \mathcal{D}$. It returns a ciphertext $c \leftarrow E(\mathsf{pk}; m)$.
- The decryption algorithm D takes as input a secret key sk and a ciphertext c, and returns a message $m \leftarrow D(\mathsf{sk}; c)$, where $m \in \mathcal{D} \cup \{\perp\}$.
- The fake encryption algorithm $\tilde{F}$ takes as input a triple $(\mathsf{pk}, \mathsf{sk}, z)$ as output by G, and outputs a "fake ciphertext" $\tilde{c} \leftarrow \tilde{F}(\mathsf{pk}, \mathsf{sk}, z)$.
- The reveal algorithm $\tilde{R}$ takes as input a triple $(\mathsf{pk}, \mathsf{sk}, z)$ as output by G, a "fake ciphertext" $\tilde{c}$ as output by $\tilde{F}$, and a message $m \in \mathcal{D}$. It outputs a "fake secret key" $\tilde{\mathsf{sk}} \leftarrow \tilde{R}(\mathsf{pk}, \mathsf{sk}, z; \tilde{c}, m)$. (Intuitively, $\tilde{\mathsf{sk}}$ is a valid-looking secret key for which $\tilde{c}$ decrypts to m.)

We make the standard correctness requirement; namely, for any $\mathsf{pk}, \mathsf{sk}, z$ output by G and any $m \in \mathcal{D}$, we have $D(\mathsf{sk}; E(\mathsf{pk}; m)) = m$.

Our definition of security requires, informally, that for any message m an adversary cannot distinguish whether it has been given a "real" encryption of m along with a "real" secret key, or a "fake" ciphertext along with a "fake" secret key under which the ciphertext decrypts to m. This should hold even when the adversary has non-adaptive access to a decryption oracle. We now give the formal definition.

Definition 2 (RNC-security) *Let $\mathcal{E} = (G, E, D, \tilde{F}, \tilde{R})$ be an RNCE scheme. We say that $\mathcal{E}$ is RNC-secure (or simply "secure") if the advantage of any* PPT *algorithm A in the game below is negligible in the security parameter k.*

1. *The key generation algorithm $G(1^k)$ is run to get $(\mathsf{pk}, \mathsf{sk}, z)$.*
2. *The algorithm A is given 1^k and pk as input, and is also given access to a decryption oracle $D(\mathsf{sk}; \cdot)$. It then outputs a challenge message $m \in \mathcal{D}$.*
3. *A bit b is chosen at random. If $b = 1$ then a ciphertext $c \leftarrow E(\mathsf{pk}; m)$ is computed, and A receives (c, sk). Otherwise, a "fake" ciphertext $\tilde{c} \leftarrow \tilde{F}(\mathsf{pk}, \mathsf{sk}, z)$ and a "fake" secret key $\tilde{\mathsf{sk}} \leftarrow \tilde{R}(\mathsf{pk}, \mathsf{sk}, z; \tilde{c}, m)$ are computed, and A receives $(\tilde{c}, \tilde{\mathsf{sk}})$. (After this point, A can no longer query its decryption oracle.) A outputs a bit b'.*

The advantage of A is defined as $2 \cdot \left|\Pr[b' = b] - \frac{1}{2}\right|$.

It is easy to see that the RNC-security of $(G, E, D, \tilde{F}, \tilde{R})$ according to Definition 2 implies in particular that the underlying scheme (G, E, D) is secure against non-adaptive chosen-ciphertext attacks. It is possible to augment Definition 2 so as to grant the adversary access to the decryption oracle even after the ciphertext is known, but we do not need this stronger definition for our intended application (Section 5). We also comment that the Naor-Yung construction [NY90] is RNC-secure for 1-bit messages (if the secret key is chosen at random from the two underlying secret keys); a proof can be derived from [NY90] as well as our proof of Theorem 1.

4.2 A Secure RNCE Scheme for Polynomial-Size Message Spaces

Here, we show that the Cramer-Shoup cryptosystem [CS98] can be modified to give a secure RNCE scheme for *polynomial-size* message spaces. Interestingly, because our definition of security only involves non-adaptive chosen-ciphertext attacks, we can base our construction on the simpler and more efficient "Cramer-Shoup lite" scheme. In fact, the only difference is that we encode a message m by the group element g^m, rather than encoding it directly as the element m. (This encoding is essential for the reveal algorithm $\tilde{R}$.[3])

In what follows, we let $\mathcal{G} = \{\mathbb{G}_k\}_{k\in\mathbb{N}}$ be a family of finite, cyclic groups (written multiplicatively), where each group $\mathbb{G}_k$ has (known) prime order q_k and $|q_k| = k$. For simplicity, we describe our RNCE scheme for the message space $\{0,1\}$; however, we will comment briefly afterward how the scheme can be extended for any polynomial-size message space.

Key generation, G**.** Given the security parameter 1^k, let $\mathbb{G}$ denote $\mathbb{G}_k$ and q denote q_k. Choose at random $g_1 \leftarrow \mathbb{G}\setminus\{1\}$, and also choose random $\alpha, x_1, x_2, y_1, y_2 \leftarrow \mathbb{Z}_q$. Set $g_2 = g_1^\alpha$; $h = g_1^{x_1} g_2^{x_2}$; and $d = g_1^{y_1} g_2^{y_2}$. The public key is $\mathsf{pk} = (g_1, g_2, h, d)$, the secret key is $\mathsf{sk} = (x_1, x_2, y_1, y_2)$, and the auxiliary information is $z = \alpha$.

Encryption, E**.** Given a public key $\mathsf{pk} = (g_1, g_2, h, d)$ and a message $m \in \{0,1\}$, choose a random $r \in \mathbb{Z}_q$, compute $u_1 = g_1^r$ $u_2 = g_2^r$, $e = g_1^m h^r$ and $v = d^r$. The ciphertext is $\langle u_1, u_2, e, v\rangle$.

Decryption, D**.** Given a ciphertext $\langle u_1, u_2, e, v\rangle$ and secret key $\mathsf{sk} = (x_1, x_2, y_1, y_2)$, proceed as follows: First check whether $u_1^{y_1} u_2^{y_2} = v$. If not, then output $\perp$. Otherwise, compute $w = e/u_1^{x_1} u_2^{x_2}$. If $w = 1$ (i.e., the group identity), output 0; if $w = g_1$, output 1. (If $w \notin \{1, g_1\}$ then output $\perp$.)

Fake encryption, $\tilde{F}$**.** Given $\mathsf{pk} = (g_1, g_2, h, d)$ and $\mathsf{sk} = (x_1, x_2, y_1, y_2)$, choose at random $r \in \mathbb{Z}_q$. Then compute $\tilde{u}_1 = g_1^r$, $\tilde{u}_2 = g_1 g_2^r$, $\tilde{e} = g_1^{x_2} h^r$ and $\tilde{v} = \tilde{u}_1^{y_1} \tilde{u}_2^{y_2}$, and output the "fake" ciphertext $\tilde{c} = \langle \tilde{u}_1, \tilde{u}_2, \tilde{e}, \tilde{v}\rangle$.

Reveal algorithm, $\tilde{R}$**.** Given $\mathsf{pk} = (g_1, g_2, h, d)$, $\mathsf{sk} = (x_1, x_2, y_1, y_2)$, $z = \alpha$, a "fake" ciphertext $\langle \tilde{u}_1, \tilde{u}_2, \tilde{e}, \tilde{v}\rangle$, and a message $m \in \{0,1\}$, set $x_2' = x_2 - m$ and $x_1' = x_1 + m\alpha$ (both in $\mathbb{Z}_q$) and output the "fake" secret key $\tilde{\mathsf{sk}} = (x_1', x_2', y_1, y_2)$.

One can check that the secret key $\tilde{\mathsf{sk}}$ matches the public key pk, since

$$g_1^{x_1'} g_2^{x_2'} \;=\; g_1^{x_1 + m\alpha} g_2^{x_2 - m} \;=\; (g_1^{x_1} g_2^m) g_2^{x_2 - m} \;=\; g_1^{x_1} g_2^{x_2} \;=\; h;$$

moreover, $\tilde{\mathsf{sk}}$ decrypts the "fake" ciphertext $\langle \tilde{u}_1, \tilde{u}_2, \tilde{e}, \tilde{v}\rangle$ to m, since

$$\frac{e}{\tilde{u}_1^{x_1'} \tilde{u}_2^{x_2'}} \;=\; \frac{g_1^{x_2} (g_1^{x_1'} g_2^{x_2'})^r}{(g_1^r)^{x_1'} (g_1 g_2^r)^{x_2'}} \;=\; \frac{g_1^{x_2 + r x_1'} g_2^{r x_2'}}{g_1^{r x_1' + x_2'} g_2^{r x_2'}} \;=\; g_1^{x_2 - x_2'} \;=\; g_1^m.$$

The above scheme can be immediately extended to support any polynomial-size message space: encryption, fake encryption, and reveal would be exactly the same, and decryption would involve computation of w, as above, followed by an exhaustive search through the message space to determine $m \stackrel{\text{def}}{=} \log_{g_1} w$.

[3]Looking ahead, it is for this reason that the present construction only handles *polynomial-size* message spaces: the receiver only directly recovers g^m, and must search through the message space to find the corresponding message m.

Before stating the formal theorem, we first recall the *decisional Diffie-Hellman (DDH) assumption*. The DDH assumption for $\mathcal{G}$ states that the following two distribution ensembles are computationally indistinguishable:

$$\mathcal{U} = \left\{ g \leftarrow \mathbb{G}_k,\ a, b, c \leftarrow \mathbb{Z}_{q_k} : (g, g^a, g^b, g^c) \right\}_{k \in \mathbb{N}} \text{ and } \mathcal{D} = \left\{ g \leftarrow \mathbb{G}_k,\ a, b \leftarrow \mathbb{Z}_{q_k} : (g, g^a, g^b, g^{ab}) \right\}_{k \in \mathbb{N}},$$

where elements drawn from the former distribution ensemble are called "random tuples" and elements drawn from the latter distribution ensemble are called "Diffie-Hellman tuples". It is easy to show that under the DDH assumption, the distribution ensemble

$$\mathcal{D}' = \left\{ g \leftarrow \mathbb{G}_k,\ a, b \leftarrow \mathbb{Z}_{q_k} : (g, g^a, g^b, g^{ab+1}) \right\}_{k \in \mathbb{N}}$$

is also indistinguishable from $\mathcal{U}$. The reason is that if the 4-tuple (g, h, u, v) is drawn at random from $\mathcal{D}$ then (g, h, u, gv) is a random tuple in $\mathcal{D}'$, whereas if (g, h, u, v) is a random tuple from $\mathcal{U}$, then so is (g, h, u, gv). It follows that $\mathcal{D}'$ is indistinguishable from $\mathcal{D}$.

Theorem 5 *If the DDH assumption holds for $\mathcal{G}$, then the above scheme is RNC-secure.*

Proof. Recall the game as specified in Definition 2. In step 3 of the game, the adversary is given either a ciphertext output by the real encryption algorithm and the "real" secret key (if $b = 1$), or a "fake" ciphertext (i.e., one output by the fake encryption algorithm) and a "fake" secret key (if $b = 0$). We show that an adversary who can distinguish these cases with a non-negligible advantage implies a distinguisher between $\mathcal{D}$ and $\mathcal{D}'$ (above), thereby violating the DDH assumption.

Fix a PPT adversary A, denote the advantage of A in the game of Definition 2 by $\epsilon = \epsilon(k)$, and let $q_d = q_d(k)$ be a polynomial upper bound on the number of decryption queries that A makes. We begin by showing that a certain "bad event" can only happen with negligible probability in this game. Let Bad denote the event that the adversary ever submits to its decryption oracle a ciphertext $c = \langle u_1, u_2, e, v \rangle$ for which (1) $u_1^\alpha \neq u_2$ (recall that $\alpha = \log_{g_1}(g_2)$), yet (2) $u_1^{y_1} u_2^{y_2} = v$. We claim that $\Pr[\mathsf{Bad}] \leq q_d/q$. (Recall that q_d is polynomial in k whereas $q = |\mathbb{G}_k|$ is exponential in k.) This is so because the public key pk only constrains the values y_1, y_2 (in an information-theoretic sense) to those q pairs satisfying $g_1^{y_1} g_2^{y_2} = d$. Furthermore, for any u_1, u_2, v for which $u_1^\alpha \neq u_2$, exactly one of those pairs satisfies $u_1^{y_1} u_2^{y_2} = v$. Other than the information in the public key, the only information available to A on y_1, y_2 comes from previous failed decryption queries, and each such query can rule out at most one pair. It follows that $\Pr[\mathsf{Bad}] \leq q_d/q$.

We next claim that, conditioned on event Bad not occurring, the answers that A gets from its decryption queries match the answers it can later compute using $D(\tilde{\mathsf{sk}}; \cdot)$, where $\tilde{\mathsf{sk}}$ is the secret key it gets in step 3 of the game. Indeed, the y_1, y_2 components in the secret key that A gets are the same as in the secret key that is used to answer the oracle queries. Furthermore, for any decryption query $\langle u_1, u_2, e, v \rangle$ that is not rejected it must be the case that $\log_{g_1} u_1 = \log_{g_2} u_2$ (assuming Bad does not occur). Letting $r = \log_{g_1} u_1$, we see that in this case the ciphertext is decrypted to (the discrete logarithm of) e/h^r regardless of (x_1, x_2), the representation of h in the secret key.

We now describe a distinguisher B between $\mathcal{D}$ and $\mathcal{D}'$. It gets a 4-tuple (g_1, g_2, u_1, u_2) and it (essentially) needs to decide whether $\log_{g_1} u_1 = \log_{g_2} u_2$ or $\log_{g_1} u_1 = \log_{g_2}(u_2/g_1)$. It chooses $x_1, x_2, y_1, y_2 \leftarrow \mathbb{Z}_q$ and gives to A the public key $(g_1, g_2,\ h = g_1^{x_1} g_2^{x_2},\ d = g_1^{y_1} g_2^{y_2})$. Then B uses its secret key to answer the decryption queries of A in step 2. When A outputs its challenge m at the end of step 2, then B computes $e = g_1^m u_1^{x_1} u_2^{x_2}$ and $v = u_1^{y_1} u_2^{y_2}$ and returns the ciphertext $c = \langle u_1, u_2, e, v \rangle$ and the secret key $\mathsf{sk} = (x_1, x_2, y_1, y_2)$. Finally, B outputs whatever A does.

It is clear that when (g_1, g_2, u_1, u_2) is a Diffie-Hellman tuple, the view of A is distributed identically to its view in the game of Definition 2 with the bit $b = 1$. We now show that when (g_1, g_2, u_1, u_2) are taken from $\mathcal{D}'$, the view of A is statistically close to its view in the game with $b = 0$: First, note that the public key pk and "fake" ciphertext c are distributed the same. Namely, g_1 is a random element in $\mathbb{G}$, g_2, d, h, e, v are random (and independent) elements in $\langle g_1 \rangle$, and u_1, u_2 are random subject to $\log_{g_1}(u_1) = \log_{g_2}(u_2/g_1)$. Next, observe that given the public key, the challenge message m, and the fake ciphertext c, there is a unique secret key that matches pk and decrypts c to m, and this is the secret key that A sees in either game. It is therefore left to show that the answers from the decryption oracles are distributed (almost) the same in both settings (i.e., even when conditioned on the secret key that is later revealed). In the run of B these answers are exactly what can be computed using $D(\mathsf{sk}; \cdot)$, where sk is the secret key that A gets in step 3 of the game. And, as explained above, A gets the same answers in the game with $b = 0$ unless event Bad occurs (which happens with negligible probability). We conclude that the advantage of B in distinguishing $\mathcal{D}$ from $\mathcal{D}'$ is at least $\epsilon - \frac{q_d}{q}$, which completes the proof. ■

4.3 A Secure RNCE Scheme for Exponential-Size Message Spaces

The RNCE scheme in the previous section can be used only for message spaces of size polynomial in the security parameter, as the decryption algorithm works in time linear in the size of the message space. We now show a scheme that supports message spaces of size exponential in the security parameter. Just as in the previous section, we construct our scheme by appropriately modifying a (standard) cryptosystem secure against chosen-ciphertext attacks. Here, we base our construction on schemes developed independently by Gennaro and Lindell [GL03] and Camenisch and Shoup [CS03], building on earlier work by Cramer and Shoup [CS02]. Security of our scheme, as in these earlier schemes, is predicated on the decisional composite residuosity (DCR) assumption [P99].

Let p, q, p', q' be distinct primes with $p = 2p' + 1$ and $q = 2q' + 1$ (i.e., p, q are *strong* primes). Let $n = pq$ and $n' = p'q'$, and observe that the group $\mathbb{Z}^*_{n^2}$ can be decomposed as the direct product $\mathbb{G}_n \cdot \mathbb{G}_{n'} \cdot \mathbb{G}_2 \cdot \mathbf{T}$, where each $\mathbb{G}_i$ is a cyclic group of order i and $\mathbf{T}$ is the order-2 subgroup of $\mathbb{Z}^*_{n^2}$ generated by $(-1 \bmod n^2)$. This implies that there exist homomorphisms $\phi_n, \phi_{n'}, \phi_2, \phi_T$ from $\mathbb{Z}^*_{n^2}$ onto $\mathbb{G}_n$, $\mathbb{G}_{n'}$, $\mathbb{G}_2$, and $\mathbf{T}$, respectively, and every $x \in \mathbb{Z}^*_{n^2}$ is uniquely represented by the 4-tuple $(\phi_n(x), \phi_{n'}(x), \phi_2(x), \phi_T(x))$. We use also the fact that the element $\gamma \stackrel{\text{def}}{=} (1 + n) \bmod n^2$ has order n in $\mathbb{Z}^*_{n^2}$ (i.e., it generates a group isomorphic to $\mathbb{G}_n$) and furthermore $\gamma^a \bmod n^2 = 1 + an$, for any $0 \le a < n$.

Let $\mathbf{P}_n \stackrel{\text{def}}{=} \{x^n \bmod n^2 : x \in \mathbb{Z}^*_{n^2}\}$ denote the subgroup of $\mathbb{Z}^*_{n^2}$ consisting of all n^{th} powers; note that $\mathbf{P}_n$ is isomorphic to the direct product $\mathbb{G}_{n'} \cdot \mathbb{G}_2 \cdot \mathbf{T}$. The DCR assumption (informally) is that, given n, it is hard to distinguish a random element of $\mathbf{P}_n$ from a random element of $\mathbb{Z}^*_{n^2}$.

Our RNCE scheme is defined below. In this description, we let $\mathcal{G}$ be an algorithm that on input 1^k randomly chooses two primes p', q' as above with $|p'| = |q'| = k$. Also, for a positive real number r we denote by $[r]$ the set $\{0, \ldots, \lfloor r \rfloor - 1\}$.

Key generation, G**.** Given the security parameter 1^k, use $\mathcal{G}(1^k)$ to select two random k-bit primes p', q' for which $p = 2p' + 1$ and $q = 2q' + 1$ are also prime, and set $n = pq$ and $n' = p'q'$. Choose random $x, y \in [n^2/4]$ and a random $g' \in \mathbb{Z}^*_{n^2}$, and compute $g = (g')^{2n}$, $h = g^x$, and $d = g^y$. The public key is $\mathsf{pk} = (n, g, h, d)$, the secret key is $\mathsf{sk} = (x, y)$, and the auxiliary information is $z = n'$.

Encryption, E**.** Given a public key as above and a message $m \in [n]$, choose random $r \in [n/4]$, compute $u = g^r$, $e = \gamma^m h^r$, and $v = d^r$ (all in $\mathbb{Z}^*_{n^2}$), and output the ciphertext $c = \langle u, e, v \rangle$.

Decryption, D. Given a ciphertext $\langle u, e, v\rangle$ and secret key (x, y), check whether $u^{2y} = v^2$; if not, output $\perp$. Then, set $\hat{m} = (e/u^x)^{n+1}$. If $\hat{m} = 1 + mn$ for some $m \in [n]$, then output m; otherwise, output $\perp$.

Correctness follows, since for a valid ciphertext $\langle u, e, v\rangle$ we have $u^{2y} = (g^r)^{2y} = d^{2r} = v^2$, and also $(e/u^x)^{n+1} = (\gamma^m h^r/g^{rx})^{n+1} = (\gamma^m)^{n+1} = \gamma^m = 1 + mn$ (using for the third equality the fact that the order of γ is n).

Fake encryption, $\tilde{F}$. Given $\mathsf{pk} = (n, g, h, d)$ and $\mathsf{sk} = (x, y)$, choose at random $r \in [n/4]$, compute $\tilde{u} = \gamma \cdot g^r$, $\tilde{e} = \tilde{u}^x$, and $\tilde{v} = \tilde{u}^y$ (all in $\mathbb{Z}^*_{n^2}$), and output the "fake" ciphertext $\tilde{c} = \langle\tilde{u}, \tilde{e}, \tilde{v}\rangle$.

Reveal algorithm, $\tilde{R}$. Given $\mathsf{pk} = (n, g, h, d)$, $\mathsf{sk} = (x, y)$, $z = n'$, a "fake" ciphertext $\langle\tilde{u}, \tilde{e}, \tilde{v}\rangle$ as above, and a message $m \in [n]$, proceed as follows: Using the Chinese Remainder Theorem and the fact that $\gcd(n, n') = 1$, find the unique $x' \in [nn']$ satisfying $x' = x \bmod n'$, and $x' = x - m \bmod n$, and output the secret key $\tilde{\mathsf{sk}} = (x', y)$.

It can be verified that the secret key $\tilde{\mathsf{sk}}$ matches the public key pk and also decrypts the "fake" ciphertext to the required message m: For the second component y this is immediate and so we focus on the first component x'. First, the order of g divides n' and so

$$g^{x'} = g^{x' \bmod n'} = g^{x \bmod n'} = g^x = h.$$

Furthermore, using also the fact that the order of γ in $\mathbb{Z}^*_{n^2}$ is n, we have

$$\left(\frac{\tilde{e}}{\tilde{u}^{x'}}\right)^{n+1} = \left(\frac{\gamma^x g^{rx}}{\gamma^{x'} g^{rx'}}\right)^{n+1} = \left(\gamma^{x-x' \bmod n}\right)^{n+1} = \gamma^m.$$

Before analyzing the protocol, we recall the DCR assumption. Let $\mathcal{G}$ be a randomized *instance generator* as described earlier. We refer the reader to [P99] for the original definition of the DCR assumption for $\mathcal{G}$, and use here the fact that this assumption implies thatthe following two distribution ensembles are computationally indistinguishable:

$$\mathcal{QR} = \left\{n \leftarrow \mathcal{G}(1^k),\ x \leftarrow \mathbb{Z}^*_{n^2} : (n, x^2)\right\}_{k\in\mathbb{N}} \text{ and } \mathcal{D} = \left\{n \leftarrow \mathcal{G}(1^k),\ x \leftarrow \mathbb{Z}^*_{n^2} : (n, x^{2n})\right\}_{k\in\mathbb{N}}.$$

It is easy to show that under the same assumption, the distribution ensemble

$$\mathcal{D}' = \left\{n \leftarrow \mathcal{G}(1^k),\ x \leftarrow \mathbb{Z}^*_{n^2} : (n, \gamma \cdot x^{2n})\right\}_{k\in\mathbb{N}}$$

is also indistinguishable from $\mathcal{QR}$, where $\gamma = (1 + n) \bmod n^2$. The reason is that if (n, y) is drawn at random from $\mathcal{D}$ then $(n, \gamma \cdot y)$ is distributed according to $\mathcal{D}'$, whereas if (n, y) is a random pair from $\mathcal{QR}$ then so is $(n, \gamma \cdot y)$. It follows that $\mathcal{D}'$ is indistinguishable from $\mathcal{D}$.

Theorem 6 *If the DCR assumption holds for $\mathcal{G}$, then the above scheme is RNC-secure.*

Proof. At a high level, the proof proceeds much like the proof of Theorem 5. We show that an adversary with a non-negligible advantage in the game of Definition 2 implies a distinguisher between $\mathcal{D}$ and $\mathcal{D}'$ above, thereby violating the DCR assumption. In what follows, we will use the fact that the uniform distributions over $[nn']$ and $[n^2/4]$ have negligible statistical difference. We also assume w.l.o.g. that $p > q$.

Fix a PPT adversary A, and let $q_d = q_d(k)$ be a polynomial upper bound on the number of decryption queries that A makes. Let Bad denote the event that the adversary ever submits to its decryption oracle a ciphertext $c = \langle u, e, v \rangle$ for which (1) $u \notin \mathbf{P}_n$, yet (2) $u^{2y} = v^2$. We claim that $\Pr[\mathsf{Bad}] \leq q_d/q$. (Recall that q_d is polynomial in k whereas $q \geq 2^k$.)

Consider a ciphertext $c = \langle u, e, v \rangle$, and let $u_n \stackrel{\text{def}}{=} \phi_n(u) \in \mathbb{G}_n$. If $u \notin \mathbf{P}_n$, then $u_n \neq 1$, so the order of u_n in $\mathbb{G}_n$ is either p, q, or n. Assume the order is q (the arguments for the other cases are similar). Since the order of g in $\mathbb{Z}^*_{n^2}$ divides n', the only possible information about y revealed by the public key pk, in an information-theoretic sense, is the value $(y \bmod n')$. In particular, given only the public key the value of $(y \bmod n)$ is still distributed uniformly in $[n]$, and so $(y \bmod q)$ is uniform in $[q]$ (recall we may make the simplifying assumption that y is chosen uniformly at random from $[nn']$).

The only other information that A has on $(y \bmod q)$, at the time it makes its decryption query c, comes from previous decryption queries $\langle u', e', v' \rangle$ where, assuming Bad did not occur, A learns only that $(u')^{2y} \neq (v')^2$. When $u' \in \mathbf{P}_n$, the order of u' in $\mathbb{Z}^*_{n^2}$ divides $2n'$ and so the condition $(u')^{2y} \stackrel{?}{=} (v')^2$ depends only on $(y \bmod n')$; hence, A does not learn anything about $(y \bmod n)$ (or $(y \bmod q)$) from such queries. When $u' \notin \mathbf{P}_n$, the fact that $(u')^{2y} \neq (v')^2$ reveals no more information on $(y \bmod q)$ than the condition $\phi_n(u')^{2y} \neq \phi_n(v')^2$ (all the computations in $\mathbb{G}_n$). Since in this case $\phi_n(u')$ has order either p, q, or n in $\mathbb{G}_n$, the condition $\phi_n(u')^{2y} \neq \phi_n(v')^2$ rules out at most one value of $(y \bmod q)$. In summary, each previous decryption query of A rules out at most one of the q potential values for $(y \bmod q)$ and hence the probability that Bad occurs on the current query is at most $1/(q - i)$, where i is the number of previous decryption queries. It follows easily that $\Pr[\mathsf{Bad}] \leq q_d/q$.

We next claim that, conditioned on event Bad not occurring, the answers that A gets from its decryption queries are independent of $(x \bmod n)$. Indeed, for a decryption query $c = \langle u, e, v \rangle$, if $u^{2y} \neq v^2$ then the query is rejected. Otherwise, assuming Bad does not occur then $u \in \mathbf{P}_n$ which implies that the order of u divides $2n'$. In this case, the value of e/u^{2x} depends only on $(x \bmod n')$, and hence it is independent of $(x \bmod n)$. It follows that, conditioned on event Bad not occurring, the answers that A gets from its decryption oracle queries match the answers it can later compute using the secret key it gets in stage 3 of the game.

We now describe a distinguisher B between $\mathcal{D}$ and $\mathcal{D}'$. It gets a pair $(n, \bar{u})$ and it (essentially) needs to decide whether $\bar{u}$ or $\bar{u}/\gamma$ are $2n^{\text{th}}$ residues in $\mathbb{Z}^*_{n^2}$. B chooses a random $g' \in \mathbb{Z}^*_{n^2}$ and random $x, y \in [n^2/4]$. It then sets $g = (g')^{2n}, h = g^x$, and $d = g^y$ and gives to A the public key (n, g, h, d). Then B uses the secret key $\mathsf{sk} = (x, y)$ to answer decryption oracle queries of A in step 2 of the game. When A outputs a message m, then B returns to A the ciphertext $\langle \bar{u}, \gamma^m \bar{u}^x, \bar{u}^y \rangle$ and secret key (x, y). Finally, B outputs whatever A does.

When B is given a tuple from $\mathcal{D}$, it is immediate that the view of A is statistically close to its view in the game of Definition 2 with $b = 1$. To see this, first note that g has order n' (both in the above game as well as the game of Definition 2) with all but negligible probability. Assuming this to be the case, we may write the first component of the ciphertext in the above game as g^r with r uniformly distributed in $[n']$, while the first component of the ciphertext in the game of Definition 2 (with $b = 1$) is g^r with r uniformly distributed in $[n/4]$. These two distributions are statistically close. The remaining components of the ciphertext, as well as those of the public key, are distributed identically in each of the two games.

We now show that when B is given a tuple from $\mathcal{D}'$, the view of A is statistically close to its view in the game of Definition 2 with $b = 0$. As before, we assume g has order n' since this occurs with all but negligible probability. In this case, we may write the first component of the ciphertext in the current game as γg^r with r uniformly distributed in $[n']$, while the first component of the ciphertext in the game of Definition 2 (with $b = 0$) is g^r with r uniformly distributed in $[n/4]$. Again, these distributions are statistically close. Finally, as long as Bad does not occur the secret key received by A exactly matches the public key, the ciphertext/message pair, and all decryption oracle answers received by A. The theorem follows. ∎

5 AFSE Based on Receiver Non-Committing Encryption

We describe a construction of an AFSE scheme based on any secure RNCE scheme and any FSE scheme secure against chosen-ciphertext attacks. Let $\mathcal{E}' = (G', E', D', \tilde{F}, \tilde{R})$ be an RNCE scheme, and let $\mathcal{E}'' = (G'', U'', E'', D'')$ be a key-evolving encryption scheme. The message space of $\mathcal{E}'$ is $\mathcal{D}$, and we assume that ciphertexts of $\mathcal{E}'$ belong to the message space of $\mathcal{E}''$. We construct a new key-evolving encryption scheme $\mathcal{E} = (G, U, E, D)$ with message space $\mathcal{D}$ as follows:

Key generation, G**.** On security parameter 1^k, run the key-generation algorithms of both schemes, setting $(\mathsf{pk}', \mathsf{sk}', z) \leftarrow G'(1^k)$ and $(\mathsf{pk}'', \mathsf{sk}''_0) \leftarrow G''(1^k)$. The public key is $(\mathsf{pk}', \mathsf{pk}'')$ and the initial secret key is $(\mathsf{sk}', \mathsf{sk}''_0)$. (The extra information z is ignored.)

Key update, U**.** The key-update operation is derived as one would expect from $\mathcal{E}''$; namely: $U(t; \mathsf{sk}', \mathsf{sk}''_t) = (\mathsf{sk}', U''(t; \mathsf{sk}''_t))$.

Encryption, E**.** To encrypt a message $m \in \mathcal{D}$ at time t, first compute $c' \leftarrow E'(\mathsf{pk}'; m)$ and then $c \leftarrow E''(\mathsf{pk}'', t; c')$. The resulting ciphertext is just c.

Decryption, D**.** To decrypt a ciphertext c, set $c' \leftarrow D''(\mathsf{sk}''_t; c)$ and then compute $m \leftarrow D'(\mathsf{sk}'; c')$.

Theorem 7 *If $\mathcal{E}'$ is RNC-secure, and if $\mathcal{E}''$ is forward-secure against chosen-ciphertext attacks, then the combined scheme given above is an AFSE scheme.*

Before proceeding to the actual proof, let us provide some informal intuition behind the proof of the above theorem. The most interesting scenario to consider is what happens upon player corruption, when the adversary obtains the secret key for the current time period t^*. We may immediately note that messages encrypted for prior time periods $t < t^*$ remain secret; this follows from the FSE encryption applied at the "outer" layer. Next, consider adaptive security for the (at most one) outstanding ciphertext which was encrypted for some time period $t \geq t^*$. Even though the adversary can "strip off" the outer later of the encryption (because the adversary now has the secret key for time period t^*), RNC security of the inner layer ensures that a simulator can open the inner ciphertext to any desired message. The main point here is that the simulator only needs to "fake" the opening of *one* inner ciphertext, and thus RNC security suffices. (Still, since the simulator does not know in advance what ciphertext it will need to open, it actually "fakes" *all* inner ciphertexts.) Chosen-ciphertext attacks are dealt with using the chosen-ciphertext security of the outer layer, as well as the definition of RNC security (where "lunch-time security" at the inner layer is sufficient). Also, we note that reversing the order of encryptions does not work: namely, using $RNCE(FSE(m))$ does not yield adaptive security, even if the RNCE scheme is fully CCA secure (cf. Section 5.1.3).

5.1 Proof of Theorem 7

To prove Theorem 7, we need to describe a simulator and prove that no environment can distinguish between interactions with the "standard real-world adversary" and the scheme $\mathcal{E}$ and interactions with the ideal-world simulator and the functionality $\mathcal{F}_{\text{AFSE}}$ (with domain $\mathcal{D}$).

5.1.1 Description of the Simulator

The simulator interacts with the functionality $\mathcal{F}_{\text{AFSE}}$ and the environment as follows:

Key Generation. Upon receiving a message $(\texttt{KeyGen}, sid, i, 1^k)$ from the $\mathcal{F}_{\text{AFSE}}$ functionality, the simulator runs the key generation of $\mathcal{E}'$ to get $(\mathsf{pk}', \mathsf{sk}', z) \leftarrow G'(1^k)$, and the key generation of $\mathcal{E}''$ to get $(\mathsf{pk}'', \mathsf{sk}''_0) \leftarrow G''(1^k)$. It returns to the functionality the public key $\mathsf{pk} = (\mathsf{pk}', \mathsf{pk}'')$. If this is the first activation then the simulator records the variables $\mathsf{pk}^* \leftarrow \mathsf{pk}$, $R^* \leftarrow i$, $t^* \leftarrow 0$, $\mathsf{sk}^* \leftarrow (\mathsf{sk}', \mathsf{sk}''_0)$, and $z^* \leftarrow z$.

Update. On a message $(\texttt{Update}, sid)$ from the functionality, the simulator uses the prescribed update algorithm U to update the secret key sk^*.

Encryption. Upon receiving a message $(\texttt{Encrypt}, sid, \mathsf{pk}, t, j)$ from the functionality (with $\mathsf{pk} = \mathsf{pk}^*$), it generates a fake ciphertext for $\mathcal{E}'$ by computing $\tilde{c} \leftarrow \tilde{F}(\mathsf{pk}', \mathsf{sk}', z)$, and then encrypts this fake ciphertext under $\mathcal{E}''$ to get $c \leftarrow E''(\mathsf{pk}'', t; \tilde{c})$. The simulator returns the ciphertext c to the functionality, and records c together with the inner ciphertext $\tilde{c}$.

Decryption. Upon receiving a message $(\texttt{Decrypt}, sid, t^*, c, i)$ from the functionality, the simulator uses the prescribed decryption algorithm D to decrypt it.

Corruptions. When the simulator is instructed by the environment to corrupt the receiver R^*, it informs the functionality that R^* is corrupted, and gets all the tuples (m, t, c, flag) with $t \geq t^*$. (Recall that there can be at most one outstanding tuple). Then:

- If there is an outstanding tuple $(m, t, c, \mathsf{outstanding})$, the simulator recovers the fake inner ciphertext $\tilde{c}$ that it recorded with the ciphertext c. It runs the reveal algorithm to get $\tilde{\mathsf{sk}} \leftarrow \tilde{R}(\mathsf{pk}', \mathsf{sk}', z^*; \tilde{c}, m)$, and re-sets $\mathsf{sk}^* \leftarrow (\tilde{sk}, \mathsf{sk}'')$. (Namely, it replaces sk' with $\tilde{\mathsf{sk}}$.)
- The simulator then returns to the environment the key $(\mathsf{pk}^*, t^*, \mathsf{sk}^*)$, and then erases all the variables R^*, pk^*, t^*, sk^*, and z^*.

5.1.2 Proof of Indistinguishability

We again use the notations and observations from Section 3.1.1. The proof consists of three parts, showing very roughly that: (1) the secrecy of encryption queries — *except the critical one* — is due to the secrecy of the outer FSE scheme $\mathcal{E}''$; (2) the secrecy of the critical encryption query is due to the secrecy of the inner RNCE $\mathcal{E}'$; and (3) the chosen-ciphertext security is due to the chosen-ciphertext security of both $\mathcal{E}'$ and $\mathcal{E}''$. However, these arguments no longer fit in a clean "simulated to hybrid" and "hybrid to real" framework. Rather, we directly show a reduction from an adversary distinguishing the simulated world from the real world to an adversary breaking the security of the outer scheme $\mathcal{E}''$, and use the other security requirements as sub-claims in the analysis of this upper-level reduction.

All in all, we describe below four PPT algorithms, A, A', B and C, where A, A' attack the forward-CCA-security of $\mathcal{E}''$ and B, C attack the RNC-security of $\mathcal{E}'$. We show that the advantage of the environment in distinguishing the simulated world from the real world is polynomially related to the sum of the advantages of these four algorithms, thereby proving:

Proposition 8 (simulated $\approx$ real) *For any* PPT *environment* Env*, the following is negligible:*

$$\alpha(k) \quad \stackrel{\text{def}}{=} \quad \left| \Pr_{\text{sim}}[\mathsf{Env}(1^k) \to 1] \;-\; \Pr_{\text{real}}[\mathsf{Env}(1^k) \to 1] \right|.$$

Proof. Fix an environment Env, and let $\alpha = \alpha(k)$ be the advantage of Env in distinguishing the simulated world from the real world. Let $T = T(k)$ be an upper bound on the local time of the verifier when it is corrupted by Env, and let $Q = Q(k)$ be an upper bound on the number of important encryption queries (pk^*, t, m) that can share the same sender time t (everything w.r.t. security parameter 1^k).

We show a chosen-ciphertext attacker A against the FSE scheme $\mathcal{E}''$ that uses Env as a subroutine. The attacker A gets as input a public key pk'', and it has access to a decryption oracle $D''(\mathsf{sk}''_{(\cdot)}; \cdot)$ (i.e., on query $(t; c)$, D'' decrypts the ciphertext c with the key sk''_t corresponding to time t). In addition, A can make one challenge query: the challenge query specifies two messages x_0, x_1 and sender time t. When A makes its challenge query, a random bit b is chosen and x_b is encrypted to get $c^* \leftarrow E''(\mathsf{pk}'', t; x_b)$. Then A gets c^* and the secret key sk''_{t+1} corresponding to time $t+1$, and its goal is to guess the bit b. A can keep making decryption queries after it makes its challenge queries, as long as it does not ask for the decryption of c^* with respect to time t.

A SLIGHT TWIST. Below, it will be more convenient to assume that A keeps running even after it outputs its guess for the bit b. Moreover, once A outputs its guess we let it grow more powerful. In particular, we now let it learn the initial secret key sk''_0 corresponding to its input public key pk''. Of course, now A can tell whether its guess is correct or not, but it cannot change its guess even if it is wrong. Clearly, this all has no effect on the advantage of A in guessing the bit b correctly, and the modified A has the same advantage as the original A. The reason for making this modification is that in the analysis of A below, we refer to some random variables whose value is only determined after A's output is already fixed.

On input pk'', the attacker A runs the environment on security parameter 1^k, implementing for it both the functionality $\mathcal{F}_{\text{AFSE}}$ and the simulator. Specifically, A answers the queries of the environment as follows:

- When the environment makes the first `KeyGen` query, A runs the key-generation of $\mathcal{E}'$ to get $(\mathsf{pk}', \mathsf{sk}', z) \leftarrow G'(1^k)$, and returns the public key $\mathsf{pk}^* = (\mathsf{pk}', \mathsf{pk}'')$. The attacker A also chooses a pair of indexes (τ, j) at random among all the pairs between $(1,1)$ and $(T,1)$. Namely

$$(\tau, j) \leftarrow \Big(\;(\{1, \ldots, T-1\} \times \{1, \ldots, Q\}) \;\;\cup\;\; \{(T,1)\}\; \Big).$$

- On an important encryption query (pk^*, t, m) with $t < \tau$, as well as on the first $j-1$ queries with $t = \tau$, A sets $c' \leftarrow E(\mathsf{pk}'; m)$. On queries with $t > \tau$ and queries (τ, j') with $j' > j$, it sets $c' \leftarrow \tilde{F}(\mathsf{pk}', \mathsf{sk}', z)$. Either way, it returns $c \leftarrow E''(\mathsf{pk}'', t; c')$.

  When it gets the j^{th} query (pk^*, t, m) with $t = \tau$, algorithm A uses its challenge query: It computes $c'_0 \leftarrow \tilde{F}(\mathsf{pk}', \mathsf{sk}', z)$ and $c'_1 \leftarrow E(\mathsf{pk}'; m)$, and makes its challenge query, specifying the "plaintexts" c'_0 and c'_1 and sender time τ. Then a bit b is chosen at random and A gets back a ciphertext $c \leftarrow E''(\mathsf{pk}'', \tau; c'_b)$ that it forwards to the environment. In addition, A also gets the secret key $\mathsf{sk}''_{\tau+1}$ for time $\tau+1$. If the local time at the receiver is still earlier than $\tau+1$ (i.e., $t^* \le \tau$), then A just stores $(\mathsf{sk}''_{\tau+1}, \tau+1)$ for later. If $t^* \ge \tau+1$ it computes the secret key sk''_{t^*} for time t^* using the update algorithm U''.

We make a slight exception for the case that A chooses the indices $\tau = T, j = 1$. In this case, A answers all the queries with time $t < T$ as above, but embeds its challenge in the first query with sender time $t \geq T$ (rather than the first query with $t = T$). Recall that there can be at most one such query, and if there is one it must be the critical query.

- On an update query, A updates the secret key sk^* if it knows it and does nothing otherwise.

- On an important decryption query (pk^*, c), A uses its own decryption oracle to decrypt c with time t^*, thus getting some $c' \leftarrow D''(\mathsf{sk}''_{t^*}; c)$. Then A uses the decryption routine of $\mathcal{E}'$ with the secret key sk' that it knows and returns $m \leftarrow D'(\mathsf{sk}'; c')$ to the environment.

- When the environment corrupts the receiver at local time t^*, the attacker A is supposed to provide the environment (among other things) with the current secret key sk''_{t^*}. We consider three cases: either A did not use its challenge query yet, or it did use its challenge query for some time $\tau < t^*$, or it used the challenge query with time $\tau \geq t^*$. In the first case A just makes up some challenge query with sender time $t^* - 1$, thus getting the key sk''_{t^*} for time t^*. In the second case, A already knows the key sk''_{t^*} for time t^*. In the third case, A has no means of getting the secret key that Env expects to see, so it just outputs zero as its guess for the bit b. However, since we assume the "modified" version of A (see the earlier remark), we allow A to now learn the initial secret key sk''_0, from which it can compute the current key sk''_{t^*} for time t^*.

  Either way, A looks up the outstanding ciphertext c and the corresponding message m (both kept by the functionality $\mathcal{F}_{\mathrm{AFSE}}$), and also the associated "inner ciphertext" $\tilde{c}$ (kept by the simulator). If sk' decrypts $\tilde{c}$ to m, then A gives the environment the key $\mathsf{sk}^* = (\mathsf{sk}', \mathsf{sk}''_{t^*})$. Otherwise, it must be that the inner ciphertext was generated as $\tilde{F}(\mathsf{pk}', \mathsf{sk}', z)$. In this case, A uses the reveal routine to get $\tilde{\mathsf{sk}} \leftarrow \tilde{R}(\mathsf{pk}', \mathsf{sk}', z^*; \tilde{c}, m)$ and returns to the environment $\mathsf{sk}^* = (\tilde{\mathsf{sk}}, \mathsf{sk}''_{t^*})$.

- When the environment halts with some output, A also halts with the same output (unless it already output zero as in the previous bullet).

We first note that when A chooses the indices $\tau = 1, j = 1$ and the bit b (that determines which of c'_0, c'_1 is encrypted in response to A's challenge query) is zero, the view of the environment is exactly as in the simulated world. Similarly, when A chooses $\tau = T, j = 1$ and the bit b is one, the view of Env is exactly as in the real world. Also, it is clear from the description of A that the view of Env when A chooses τ, j and $b = 1$ is identical to its view when A chooses $\tau, j+1$ and $b = 0$.[4] This means that in the run of A we have:

$$
\begin{aligned}
&\left|\Pr_A[\mathsf{Env} \to 1 \mid b = 1] - \Pr_A[\mathsf{Env} \to 1 \mid b = 0]\right| \\
&= \left|\frac{1}{Q(T-1)+1} \sum_{\tau,j} \left(\Pr_A[\mathsf{Env} \to 1 \mid \tau, j, b = 1] - \Pr_A[\mathsf{Env} \to 1 \mid \tau, j, b = 0]\right)\right| \\
&= \left|\frac{1}{Q(T-1)+1} \left(\Pr_A[\mathsf{Env} \to 1 \mid \tau = T, j = 1, b = 1] - \Pr_A[\mathsf{Env} \to 1 \mid \tau = 1, j = 1, b = 0]\right)\right| \\
&> \alpha/TQ.
\end{aligned}
$$

[4]For convenience we use here $(\tau, Q+1)$ as a shorthand for $(\tau+1, 1)$, and we assume that $(\tau, j) < (T, 1)$.

This, however, still falls short of what we need, since we need to prove a similar statement for the probability that A outputs one (and A does not always output the same thing as Env). We observe that A outputs 1 whenever the environment does, except if it happened to embed its challenge query in an encryption query with sender time τ and the local time at the receiver when it is corrupted is $t^* \leq \tau$. In this case, the query in which A embedded its challenge must be the critical query, as it is still outstanding when the receiver is corrupted. In turn, this means that it must be the only important encryption query with sender time τ (in particular, it is the first query with this sender time and thus A chose $j = 1$). It also means that all the other important encryption queries had sender time $t < t^* \leq \tau$.

Below we denote by $\mathsf{Critical}$ the event in which A embeds its challenge in the critical encryption query. Namely, this is the event that A chooses $j = 1$ and some $\tau < T$ and the critical query has sender time τ, or when A chooses $j = 1$ and $\tau = T$ and the critical query has sender time $t \geq T$. Using this notation, we can write

$$A \to 1 \iff \mathsf{Env} \to 1 \text{ and } \neg\mathsf{Critical}. \tag{1}$$

The key observation here is that due to the properties of the inner RNCE, when the "bad event" $\mathsf{Critical}$ occurs the environment has only a negligible advantage in distinguishing $b = 1$ from $b = 0$. Thus, it must be that (almost) all the advantage of Env comes from the "good event" (i.e., when $\mathsf{Critical}$ does not occur), and so (informally) almost all this advantage is shared by A. Formally, we have the following:

Claim 9 *Denote* $\delta \stackrel{\text{def}}{=} |\Pr_A[\mathsf{Env} \to 1 \text{ and } \mathsf{Critical} \mid b = 1] - \Pr_A[\mathsf{Env} \to 1 \text{ and } \mathsf{Critical} \mid b = 0]|$. *Then* δ *is negligible in the security parameter* k.

Obviously, proving Claim 9 is sufficient to complete the proof of Proposition 8, since

$$\begin{aligned}
\mathsf{advantage(A)} &= \left|\Pr_A[A \to 1 \mid b = 1] - \Pr_A[A \to 1 \mid b = 0]\right| \\
&= \left|\Pr_A[\mathsf{Env} \to 1 \text{ and } \neg\mathsf{Critical} \mid b = 1] - \Pr_A[\mathsf{Env} \to 1 \text{ and } \neg\mathsf{Critical} \mid b = 0]\right| \\
&\geq \left|\Pr_A[\mathsf{Env} \to 1 \mid b = 1] - \Pr_A[\mathsf{Env} \to 1 \mid b = 0]\right| - \delta \\
&\geq \alpha/TQ - \delta.
\end{aligned}$$

The CCA-security of $\mathcal{E}''$ ensures that the advantage of A is negligible, and as T, Q are polynomially bounded and δ is negligible we may thus conclude that α is also negligible. ■

Proof of Claim 9 For the most part, the claim follows from Definition 2: When A embeds its challenge in the critical encryption query, the only difference in the view of the environment in the two cases $b = 0$ and $b = 1$, is that in the first case the inner ciphertext in the critical query is as in the real world (i.e., c' is generated according to $c' \leftarrow E'(\mathsf{pk}'; m)$) and Env gets the secret key sk', while in the second case, the inner ciphertext is as in the simulated world (i.e., c' is generated using $c' \leftarrow \tilde{F}(\mathsf{pk}', \mathsf{sk}', z)$) and Env gets a secret key $\tilde{\mathsf{sk}}$ that decrypts c' to m. By Definition 2, Env cannot distinguish between these two cases.

The only subtlety here is that the view of Env is not the quite same as that of the distinguisher in Definition 2: The experiment in Definition 2 describes a "lunch-time attack" scenario in which the attacker cannot query its decryption oracle after it gets the “target ciphertext”. On the other hand, Env can keep making decryption queries, even after it asks the critical decryption query. However, we show that this discrepancy does not help Env to distinguish $b = 0$ from $b = 1$.

We describe an algorithm B for the experiment from Definition 2. The algorithm B is designed so that the view of the environment is the same as in the run of A up until the event $\mathsf{Critical}$ is decided, and “almost the same” as in the run of A conditioned on this event occurring. Moreover, B outputs one exactly when Env outputs one and the event $\mathsf{Critical}$ happens. This implies that B has advantage ϵ, which is “almost the same” as δ from Claim 9. The RNC-security of $\mathcal{E}'$ ensures that ϵ is negligible, so we conclude that δ must also be negligible.

On input pk', the algorithm B chooses (τ, j) at random in $([1..T-1] \times [1..Q]) \cup \{(T, 1)\}$, just as A does. If $j \neq 1$ then B immediately halts, outputting zero. Otherwise, B runs the environment on security parameter 1^k. When the environment makes its first $\mathtt{KeyGen}$ query, B chooses the keys for the outer encryption $\mathcal{E}''$ by running $(\mathsf{pk}'', \mathsf{sk}''_0) \leftarrow G''(1^k)$, and returns $\mathsf{pk}^* = (\mathsf{pk}', \mathsf{pk}'')$. After that, B answers the environment queries as follows: On important encryption queries (pk^*, t, m) with $t < \tau$, B answers as in the real world, setting $c \leftarrow E''(\mathsf{pk}'', t; E'(\mathsf{pk}'; m))$. If the environment makes important encryption queries with sender time $t > \tau$, or if it makes more than one important query with sender time $t = \tau$, then B outputs zero and halts. If and when the environment makes its first query (pk^*, τ, m) with sender time $t = \tau$, B outputs its challenge message m, getting a ciphertext $\tilde{c}$ (which is computed using either $\tilde{c} \leftarrow \tilde{F}(\mathsf{pk}', \mathsf{sk}', z)$ or $\tilde{c} \leftarrow E'(\mathsf{pk}'; m)$), and a secret key $\tilde{\mathsf{sk}}$ (that decrypts $\tilde{c}$ to m).

On update queries, B just updates the secret key sk'' of the outer encryption scheme. As long as B still did not use its challenge message, it answers important decryption queries (pk^*, c) by decrypting $c' \leftarrow D''(\mathsf{sk}''; c)$ and then using its decryption oracle $D'(\mathsf{sk}'; \cdot)$ to get $m \leftarrow D'(\mathsf{sk}; c')$. After using its challenge message, B cannot use its decryption oracle anymore so instead it uses the secret key $\tilde{\mathsf{sk}}$ that it received, answering a query (pk^*, c) with $m \leftarrow D'(\tilde{\mathsf{sk}}; D''(\mathsf{sk}''; c))$. When the environment corrupts the receiver, B checks to see if the event $\mathsf{Critical}$ happened (i.e., if the critical encryption query had time τ). If not, B outputs zero and halts. If $\mathsf{Critical}$ did happen, then B gives the environment the secret key $\mathsf{sk}^* = (\mathsf{sk}'', \tilde{\mathsf{sk}})$ and eventually outputs whatever Env does.

We denote by σ the bit chosen in the experiment of Definition 2 (which B is trying to guess). It is not hard to see that the view of Env in the run of B, conditioned on $\mathsf{Critical}$ occurring and $\sigma = 1$, is distributed exactly as its view in the run of A from above conditioned on $\mathsf{Critical}$ and $b = 1$: in both cases, all queries of Env are answered as in the real world. Also, by construction of B, the probability of $\mathsf{Critical}$ is the same in the runs of A and B (even when conditioned on $b = 1$ and $\sigma = 1$, respectively). We thus have:

$$\Pr_B[B \to 1 \mid \sigma = 1] \;=\; \Pr_B[\mathsf{Env} \to 1 \text{ and } \mathsf{Critical} \mid \sigma = 1] \;=\; \Pr_A[\mathsf{Env} \to 1 \text{ and } \mathsf{Critical} \mid b = 1]. \quad (2)$$

We would like to say the same thing about the events $\sigma = 0$ in the run of B vs. $b = 0$ in the run of A, since in both these cases all the encryption queries of the environment are answered as in the simulated world. Here, however, the views of Env may differ, since after Env makes the critical encryption query (i.e., the first and only query (pk^*, t, m) with sender time $t = \tau$) further *decryption queries* in the run of B are answered using $D'(\tilde{\mathsf{sk}}; \cdot)$, whereas in the run of A they are answered using $D'(\mathsf{sk}'; \cdot)$. However, it is sufficient to show that (with all but a negligible probability) all these decryption queries have the same answers no matter what secret key is used to answer them.

In a run of either A or B, we say that an important decryption query of Env is *ambivalent* if the inner ciphertext in that query is decrypted differently by the secret key sk' that is used at the time, than by the secret key $\tilde{\mathsf{sk}}$ that is later given to Env. We denote by $\mathsf{Ambivalent}$ the event in which Env makes an ambivalent decryption query. Clearly, as long as this does not happen the view of Env in the run of B with $\sigma = 0$ and $\mathsf{Critical}$ is distributed the same as its view in the run of A with $b = 0$ and $\mathsf{Critical}$. Hence we have

$$\begin{aligned}
&\Pr_A[\mathsf{Env} \to 1 \text{ and } \mathsf{Critical} \mid b = 0] \\
&= \Pr_A[\mathsf{Env} \to 1 \text{ and } \mathsf{Critical} \text{ and } \mathsf{Ambivalent} \mid b = 0] \\
&\quad + \Pr_A[\mathsf{Env} \to 1 \text{ and } \mathsf{Critical} \text{ and } \neg\mathsf{Ambivalent} \mid b = 0] \\
&\le \Pr_A[\mathsf{Critical} \text{ and } \mathsf{Ambivalent} \mid b = 0] \ + \ \Pr_B[\mathsf{Env} \to 1 \text{ and } \mathsf{Critical} \text{ and } \neg\mathsf{Ambivalent} \mid \sigma = 0] \\
&= \Pr_A[\mathsf{Critical} \text{ and } \mathsf{Ambivalent} \mid b = 0] \ + \ \Pr_B[B \to 1 \text{ and } \neg\mathsf{Ambivalent} \mid \sigma = 0] \\
&\le \Pr_A[\mathsf{Critical} \text{ and } \mathsf{Ambivalent} \mid b = 0] \ + \ \Pr_B[B \to 1 \mid \sigma = 0];
\end{aligned}$$

therefore:

$$\begin{aligned}
\delta \ &= \ \left| \Pr_A[\mathsf{Env} \to 1 \text{ and } \mathsf{Critical} \mid b = 1] \ - \ \Pr_A[\mathsf{Env} \to 1 \text{ and } \mathsf{Critical} \mid b = 0] \right| \\
&\le \ \Pr_A[\mathsf{Critical} \text{ and } \mathsf{Ambivalent} \mid b = 0] \ + \ \left| \Pr_B[B \to 1 \mid \sigma = 0] \ - \ \Pr_B[B \to 1 \mid \sigma = 1] \right| \\
&= \ \Pr_A[\mathsf{Critical} \text{ and } \mathsf{Ambivalent} \mid b = 0] \ + \ \epsilon,
\end{aligned}$$

where ϵ is the advantage of B. The RNC-security of $\mathcal{E}'$ implies that ϵ is negligible, so to complete the proof of Claim 9 it is sufficient to prove the following:

Claim 10 *The probability* $\Pr_A[\mathsf{Critical} \text{ and } \mathsf{Ambivalent} \mid b = 0]$ *is negligible in* k.

Proof. Roughly, we show that the event "$\mathsf{Critical}$ and $\mathsf{Ambivalent}$" does not happen in a run of A with $b = 1$,[5] due to the RNC-security of the inner scheme $\mathcal{E}'$. Therefore, it also cannot happen in a run of A with $b = 0$ or else we could use it to distinguish $b = 0$ from $b = 1$, thereby breaking the CCA-security of the outer scheme $\mathcal{E}''$.

Formally, consider the following modification of the algorithm A from above, which we denote A': As long as the receiver is not corrupted, the algorithm A' interacts with the environment Env the same way as A, and it remembers the "inner ciphertexts" in all the important decryption queries that Env makes. However, just as the algorithm B from above, A' halts with an output of zero whenever it becomes clear that the event $\mathsf{Critical}$ does not happen (i.e., if $j > 1$, or if the environment makes an important encryption query with sender time $t > \tau$, or if it makes more than one query with sender time $t = \tau$).

When the environment corrupts the receiver, A' checks once again to see if the event $\mathsf{Critical}$ happened (i.e., if the query in which it embedded its challenge happened to be the critical encryption query). If not, then A' simply outputs zero and halts. If the event $\mathsf{Critical}$ did happen, then A' recalls the message m from the critical encryption query and the two ciphertexts c'_0 and c'_1 that it used for its own challenge. (Recall that c'_0 was computed as $F(\mathsf{pk}', \mathsf{sk}', z)$ and c'_1 was computed as $E'(\mathsf{pk}'; m)$.) It computes $\tilde{\mathsf{sk}} \leftarrow \tilde{R}(\mathsf{pk}', \mathsf{sk}', z; c'_0, m)$ and then examines the inner ciphertexts in all the important decryption queries that Env made. If it finds an ambivalent inner ciphertext (i.e., a ciphertext $\tilde{c}$ such that $D'(\mathsf{sk}'; \tilde{c}) \neq D'(\tilde{\mathsf{sk}}; \tilde{c})$), then it outputs "1". Otherwise, it outputs "0".

[5] Technically, $\mathsf{Ambivalent}$ cannot happen when $b = 1$ since in this case we have $\tilde{\mathsf{sk}} = \mathsf{sk}'$, but we will make clear exactly what we mean in the actual description below.

We observe that if $b = 0$, then the key $\tilde{\mathsf{sk}}$ that A' computes is indeed the one that A would return to Env. Hence, in this case we have A' outputting one if and only if the event "$\mathsf{Critical\ and\ Ambivalent}$" happens. Namely $\Pr_{A'}[A' \to 1 | b = 0] \;=\; \Pr_A[\mathsf{Critical\ and\ Ambivalent} \mid b = 0]$. Now we can write,

$$\begin{aligned}
\Pr_A[\mathsf{Critical\ and\ Ambivalent} \mid b = 0] &= \Pr_{A'}[A' \to 1 | b = 0] \\
&\leq \Pr_{A'}[A' \to 1 | b = 1] \;+\; \delta',
\end{aligned}$$

where δ' is the advantage of A', namely $\delta' \stackrel{\text{def}}{=} |\Pr_{A'}[A' \to 1 | b = 1] \;-\; \Pr_{A'}[A' \to 1 | b = 0]|$. The CCA-security of $\mathcal{E}''$ ensures that δ' is negligible in k. To prove Claim 10, it is therefore sufficient to show that

$$\rho \stackrel{\text{def}}{=} \Pr_{A'}[A' \to 1 \mid b = 1]$$

is also negligible in k. To so so, we describe one last algorithm C, for the experiment from Definition 2 using the inner scheme $\mathcal{E}'$, and show that C has advantage exactly ρ. RNC security of $\mathcal{E}'$ then implies that ρ is negligible.

The algorithm C is designed to mimic the interaction between A' and Env for the case $b = 1$. On input pk', the algorithm C chooses (τ, j) at random just as A' does and halts with output zero if $j > 1$. Otherwise C runs the environment on security parameter 1^k. When the environment makes its first `KeyGen` query, C chooses the keys for the outer encryption $\mathcal{E}''$ by computing $(\mathsf{pk}'', \mathsf{sk}''_0) \leftarrow G''(1^k)$, and returns $\mathsf{pk}^* = (\mathsf{pk}', \mathsf{pk}'')$.

After that, C answers encryption queries (pk^*, t, m) with sender time $t < \tau$ just as A' does, returning $E''(\mathsf{pk}'', t; E'(\mathsf{pk}'; m))$. Moreover, C answers in the same way also the first encryption query with sender time τ. We note that this is indeed consistent with what A' does in the case $b = 1$ (since A' sets $c'_1 \leftarrow E(\mathsf{pk}'; m)$ in its challenge). We also note that A' halts with zero whenever Env makes a query with sender time $t > \tau$ (or more than one query with sender time $t = \tau$), so C never needs to answer a query using the fake encryption algorithm $\tilde{F}$. (Indeed, C cannot use $\tilde{F}$ in a consistent manner, since it does not know sk' and z.)

On update queries, C updates the outer secret key sk'' just as A' does. The answers it gives to decryption queries are also the same as those given by A' (since A' has oracle access to $D''(\mathsf{sk}''; \cdot)$ and it knows the inner secret key sk', whereas C knows the outer secret key sk'' and it has oracle access to $D'(\mathsf{sk}'; \cdot)$). Finally, when the environment corrupts the receiver C checks once more that the event $\mathsf{Critical}$ happened and halts with output "0" if not. If $\mathsf{Critical}$ did happen, then C recalls the message m from the critical encryption query and lets m be its challenge message. It gets back a secret key $\tilde{\mathsf{sk}}$ (and also a ciphertext c'_0 that it ignores). Using $\tilde{\mathsf{sk}}$, C makes the same test as A'; namely, it examines the inner ciphertexts from all the important decryption queries of Env, looking for a ciphertext $\tilde{c}$ such that $D'(\tilde{\mathsf{sk}}; \tilde{c})$ is not the same as what C got at the time from its decryption oracle $D'(\mathsf{sk}'; \cdot)$. If it find such an ambivalent inner ciphertext, then C outputs "1", and otherwise it outputs "0".

We again let σ denote the bit chosen during the experiment of Definition 2 which C is trying to guess. We observe that when $\sigma = 1$, then C gets the inner secret key $\tilde{\mathsf{sk}} = \mathsf{sk}'$ and therefore in this case it never finds an "ambivalent" ciphertext and it always outputs zero. Namely, $\Pr_C[C \to 1 | \sigma = 1] = 0$. On the other hand, the behavior of C when $\sigma = 0$ is identical to the behavior of A' when $b = 1$. (The interaction with Env is the same in both cases, and $\tilde{\mathsf{sk}}$ is drawn from the same distribution in both cases, so the output distribution must also be the same.) That is, $\Pr_C[C \to 1 | \sigma = 0] = \Pr_{A'}[A' \to 1 | b = 1] = \rho$. We conclude that the advantage of C is exactly ρ, so ρ must be negligible in k.

This completes the proof of Claim 10 and therefore also the proofs of Claim 9, Proposition 8, and Theorem 7. ■

5.1.3 Comments

After seeing the proof of Theorem 7, we can now explain why reversing the order between the FSE and RNCE schemes does not work. Note that in the simulated world, the simulator produces many "fake ciphertexts", whereas the definition of RNC-security only refers to a single fake ciphertext. The reason that requiring security for just one fake ciphertext is sufficient even in a world where there are many fake ciphertexts is that they are all "hidden" from the environment due to the outer layer of forward-secure encryption. The only inner ciphertext that the environment sees is the one in the critical ciphertext, and for that one we can rely on the RNC-security of the inner scheme.

If we were computing the ciphertexts as $c = RNCE(FSE(m))$, then in the simulated world the environment would potentially see many fake ciphertexts and the notion of RNC-security from Definition 2 would be insufficient. We stress that in that case, even augmenting Definition 2 to imply full CCA security (as opposed to "lunchtime security") would not be enough.

References

[AF04] M. Abe and S. Fehr. Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography. *Adv. in Cryptology — Crypto 2004*, LNCS vol. 3152, Springer-Verlag, pp. 317–334, 2004. Full version available at `eprint.iacr.org/2004/119`.

[A97] R. Anderson. Two Remarks on Public Key Cryptology. Invited lecture, given at *ACM CCCS '97*. Available at `http://www.cl.cam.ac.uk/ftp/users/rja14/forwardsecure.pdf`.

[B97] D. Beaver. Plug and Play Encryption. *Adv. in Cryptology — Crypto 1997*, LNCS vol. 1294, Springer-Verlag, pp. 75–89, 1997.

[BH92] D. Beaver and S. Haber. Cryptographic Protocols Provably Secure Against Dynamic Adversaries. *Adv. in Cryptology — Eurocrypt 1992*, LNCS vol. 658, Springer-Verlag, pp. 307–323, 1992.

[BDPR98] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among Notions of Security for Public-Key Encryption Schemes. *Adv. in Cryptology — Crypto 1998*, LNCS vol. 1462, Springer-Verlag, pp. 26–45, 1998.

[BB04] D. Boneh and X. Boyen. Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles. *Adv. in Cryptology — Eurocrypt 2004*, LNCS vol. 3027, Springer-Verlag, pp. 223–238, 2004.

[CS03] J. Camenisch and V. Shoup. Practical Verifiable Encryption and Decryption of Discrete Logarithms. *Adv. in Cryptology — Crypto 2003*, LNCS vol. 2729, Springer-Verlag, pp. 126–144, 2003.

[C01] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. *42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, IEEE, pp. 136–145, 2001. Also available as ECCC TR 01-16, or from `http://eprint.iacr.org/2000/067`.

[CFGN96] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively Secure Computation. *28th ACM Symposium on Theory of Computing (STOC)*, ACM, pp. 639–648, 1996. Full version in MIT-LCS-TR #682, 1996.

[CHK03] R. Canetti, S. Halevi, and J. Katz. A Forward-Secure Public-Key Encryption Scheme. *Adv. in Cryptology — Eurocrypt 2003*, LNCS vol. 2656, Springer-Verlag, pp. 255–271, 2003. Full version available at `http://eprint.iacr.org/2003/083`.

[CHK04] R. Canetti, S. Halevi, and J. Katz. Chosen-Ciphertext Security from Identity-Based Encryption. *Adv. in Cryptology — Eurocrypt 2004*, LNCS vol. 3027, Springer-Verlag, pp. 207–222, 2004. Full version available at `http://eprint.iacr.org/2003/182`.

[CKN03] R. Canetti, H. Krawczyk, and J.B. Nielsen. Relaxing Chosen Ciphertext Security. *Adv. in Cryptology — Crypto 2003*, LNCS vol. 2729, Springer-Verlag, pp. 565–582, 2003. Full version available at `http://eprint.iacr.org/2003/174`.

[CS98] R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure Against Chosen Ciphertext Attack. *Adv. in Cryptology — Crypto 1998*, LNCS vol. 1462, Springer-Verlag, pp. 13–25, 1998.

[CS02] R. Cramer and V. Shoup. Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. *Adv. in Cryptology — Eurocrypt 2001*, LNCS vol. 2332, Springer-Verlag, pp. 45–63, 2001.

[DN00] I. Damgård and J. B. Nielsen. Improved Non-Committing Encryption Schemes Based on General Complexity Assumptions. *Adv. in Cryptology — Crypto 2000*, LNCS vol. 1880, Springer-Verlag, pp. 432–450, 2000.

[DDOPS01] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust Non-Interactive Zero Knowledge. *Adv. in Cryptology — Crypto 2001*, LNCS vol. 2139, Springer-Verlag, pp. 566–598, 2001.

[DDN00] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. *SIAM. J. Computing* 30(2): 391-437, 2000.

[GL03] R. Gennaro and Y. Lindell. A Framework for Password-Based Authenticated Key Exchange. *Adv. in Cryptology — Eurocrypt 2003*, LNCS vol. 2656, Springer-Verlag, pp. 524–543, 2003. Full version available at `http://eprint.iacr.org/2003/032`.

[GM84] S. Goldwasser and S. Micali. Probabilistic Encryption. *J. Computer System Sciences* 28(2): 270-299, 1984.

[HMS03] Dennis Hofheinz, Joern Mueller-Quade, and Rainer Steinwandt. On Modeling IND-CCA Security in Cryptographic Protocols. Available at `http://eprint.iacr.org/2003/024`.

[JL00] S. Jarecki and A. Lysyanskaya. Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures. *Adv. in Cryptology — Eurocrypt 2000*, LNCS vol. 1807, Springer-Verlag, pp. 221–242, 2000.

[NY90] M. Naor and M. Yung. Public-Key Cryptosystems Provably-Secure against Chosen-Ciphertext Attacks. *22nd ACM Symposium on Theory of Computing (STOC)*, ACM, pp. 427–437, 1990.

[N02] J.B. Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-Committing Encryption Case. *Adv. in Cryptology — Crypto 2002*, LNCS vol. 2442, Springer-Verlag, pp. 111–126, 2002.

[P99] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. *Adv. in Cryptology — Eurocrypt 1999*, LNCS vol. 1592, Springer-Verlag, pp. 223–238, 1999.

[RS91] C. Rackoff and D. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. *Adv. in Cryptology — Crypto 1991*, LNCS vol. 576, Springer-Verlag, pp. 433–444, 1991.

[S99] A. Sahai. Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security. *40th IEEE Symposium on Foundations of Computer Science (FOCS)*, IEEE, pp. 543–553, 1999.

A Key-Evolving and Forward-Secure Encryption

We review the definitions of key-evolving and forward-secure encryption schemes from [CHK03].

Definition 3 *A (public-key) key-evolving encryption (*ke-PKE*)* scheme *is a 4-tuple of* PPT *algorithms* (Gen, Upd, Enc, Dec) *such that:*

- *The* key generation algorithm Gen *takes as input a security parameter* 1^k *and the total number of time periods* N*. It returns a public key* pk *and an initial secret key* sk_0*.*
- *The* key update algorithm Upd *takes as input* pk*, an index* $t < N$ *of the current time period, and the associated secret key* sk_t*. It returns the secret key* sk_{t+1} *for the following time period.*
- *The* encryption algorithm Enc *takes as input* pk*, an index* $t \leq N$ *of a time period, and a message* M*. It returns a ciphertext* C*.*
- *The* decryption algorithm Dec *takes as input* pk*, an index* $t \leq N$ *of the current time period, the associated secret key* sk_t*, and a ciphertext* C*. It returns a message* M*.*

We require that $\mathsf{Dec}(\mathsf{sk}_t; t; \mathsf{Enc}(\mathsf{pk}_t, t, M)) = M$ *holds for all* $(\mathsf{pk}, \mathsf{sk}_0)$ *output by* Gen*, all time periods* $t \leq N$*, all correctly generated* sk_t *for this* t*, and all messages* M*.*

Definition 4 *A ke-PKE scheme is* forward-secure against chosen plaintext attacks *(*fs-CPA*) if for all polynomially-bounded functions* $N(\cdot)$*, the advantage of any* PPT *adversary in the following game is negligible in the security parameter:*

Setup*:* $\mathsf{Gen}(1^k, N(k))$ *outputs* (PK, SK_0)*. The adversary is given* PK*.*

Attack*: The adversary issues one* $\mathsf{breakin}(i)$ *query and one* $\mathsf{challenge}(j, M_0, M_1)$ *query, in either order, subject to* $0 \leq j < i < N$*. These queries are answered as follows:*

- *On query* $\mathsf{breakin}(i)$*, key* SK_i *is computed via* $\mathsf{Upd}(PK, i-1, \cdots \mathsf{Upd}(PK, 0, SK_0) \cdots)$*. This key is then given to the adversary.*
- *On query* $\mathsf{challenge}(j, M_0, M_1)$*, a random bit* b *is selected and the adversary is given* $C^* = \mathsf{Enc}(PK, j, M_b)$*.*

Guess: *The adversary outputs a guess* $b' \in \{0,1\}$*; it* succeeds *if* $b' = b$*. The adversary's* advantage *is the absolute value of the difference between its success probability and* $1/2$*.*

Forward security against (adaptive) chosen-ciphertext attacks (fs-CCA security) is defined by the natural extension of the above definition in which the adversary is given decryption oracle access during both the "Attack" and "Guess" stages.

B The UC Framework, Abridged

We provide a brief review of the universally composable security framework [C01]. The framework allows for defining the security properties of cryptographic tasks so that security is maintained under general composition with an unbounded number of instances of arbitrary protocols running concurrently. Definitions of security in this framework are called universally composable (UC).

In the UC framework, the security requirements of a given task (i.e., the functionality expected from a protocol that carries out the task) are captured via a set of instructions for a "trusted party" that obtains the inputs of the participants and provides them with the desired outputs (in one or more iterations). Informally, a protocol securely carries out a given task if running the protocol with a realistic adversary amounts to "emulating" an ideal process where the parties hand their inputs to a trusted party with the appropriate functionality and obtain their outputs from it, without any other interaction.

The notion of emulation in the UC framework is considerably stronger than that considered in previous models. Traditionally, the model of computation includes the parties running the protocol and an adversary $\mathcal{A}$ that controls the communication channels and potentially corrupts parties. "Emulating an ideal process" means that for any adversary $\mathcal{A}$ there should exist an "ideal process adversary" (or simulator) $\mathcal{S}$ that causes the *outputs* of the parties in the ideal process to have similar distribution to the outputs of the parties in an execution of the protocol. In the UC framework the requirement on $\mathcal{S}$ is more stringent. Specifically, an additional entity, called the environment $\mathcal{Z}$, is introduced. The environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. A protocol is said to securely realize functionality $\mathcal{F}$ if for any "real-life" adversary $\mathcal{A}$ that interacts with the protocol and the environment there exists an "ideal-process adversary" $\mathcal{S}$, such that *no environment* $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties that interact with $\mathcal{F}$ in the ideal process. In a sense, $\mathcal{Z}$ serves as an "interactive distinguisher" between a run of the protocol and the ideal process with access to $\mathcal{F}$.

The following *universal composition theorem* is proven in [C01]. Consider a protocol π that operates in the $\mathcal{F}$-hybrid model, where parties can communicate as usual and in addition have ideal access to an unbounded number of *copies* of the functionality $\mathcal{F}$. Let ρ be a protocol that securely realizes $\mathcal{F}$ as sketched above, and let π^ρ be identical to π with the exception that the interaction with *each copy* of $\mathcal{F}$ is replaced with an interaction with a *separate instance* of ρ. Then, π and π^ρ have essentially the same input/output behavior. In particular, if π securely realizes some functionality $\mathcal{I}$ in the $\mathcal{F}$-hybrid model then π^ρ securely realizes $\mathcal{I}$ in the standard model (i.e., without access to any functionality).

B.1 The Public-Key Encryption Functionality $\mathcal{F}_{\text{PKE}}$

(This section is taken almost verbatim from [CKN03].) Within the UC framework, public-key encryption is defined via the public-key encryption functionality, denoted $\mathcal{F}_{\text{PKE}}$ and presented in

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

$\mathcal{F}_{\mathrm{PKE}}$ proceeds as follows, when parameterized by message domain ensemble $\mathcal{D} = \{D_k\}_{k\in\mathbf{N}}$ and security parameter k.

Key Generation: Upon receiving a value $(\texttt{KeyGen}, sid)$ from some party R^*, verify that $sid = (sid', R^*)$. If not, then ignore the input. Otherwise:

1. Hand $(\texttt{KeyGen}, sid)$ to the adversary.
2. Receive a value pk^* from the adversary, and hand pk^* to R^*.
3. If this is the first KeyGen request, record R^* and pk^*.

Encryption: Upon receiving from some party P a value $(\texttt{Encrypt}, sid, \mathsf{pk}, m)$ proceed as follows:

1. If $m \notin D_k$ then return an error message to P.
2. If $m \in D_k$ then hand $(\texttt{Encrypt}, sid, \mathsf{pk}, P)$ to the adversary. (If $\mathsf{pk} \neq \mathsf{pk}^*$ or pk^* is not yet defined then hand also the entire value m to the adversary.)
3. Receive a "ciphertext" c from the adversary, record the pair (c, m), and send $(\texttt{ciphertext,}c)$ to P. (If $\mathsf{pk} \neq \mathsf{pk}^*$ or pk^* is not yet defined then do *not* record the pair (c, m).)

Decryption: Upon receiving a value $(\texttt{Decrypt}, sid, c)$ from R^* (and R^* only), proceed as follows:

1. If there is a recorded pair (c, m) then hand m to R^*. (If there is more than one such pair then use the first one.)
2. Otherwise, hand the value $(\texttt{Decrypt}, sid, c)$ to the adversary. When receiving a value m' from the adversary, hand m' to R^*.

Figure 2: The public-key encryption functionality, $\mathcal{F}_{\mathrm{PKE}}$

Figure 2. Functionality $\mathcal{F}_{\mathrm{PKE}}$ is intended to capture the functionality of public-key encryption and, in particular, is written in a way that allows realizations consisting of three non-interactive algorithms without any communication. (The three algorithms correspond to the key generation, encryption, and decryption algorithms in traditional definitions.)

Referring to Figure 2, we note that *sid* serves as a unique identifier for an instance of functionality $\mathcal{F}_{\mathrm{PKE}}$ (this is needed in a general protocol setting when this functionality can be composed with other components, or even with other instances of $\mathcal{F}_{\mathrm{PKE}}$). It also encodes the identity of the decryptor for this instance. The "public key value" pk has no particular meaning in the ideal scenario beyond serving as an identifier for the public key related to this instance of the functionality, and this value can be chosen arbitrarily by the attacker. Also, in the ideal setting ciphertexts serve as identifiers or tags with no particular relation to the encrypted messages (and as such are also chosen by the adversary without knowledge of the plaintext). Still, rule 1 of the decryption operation guarantees that "legitimate ciphertexts" (i.e., those produced and recorded by the functionality under an Encrypt request) are decrypted correctly, while the resultant plaintexts remain unknown to the adversary. In contrast, ciphertexts that were not legitimately generated can be decrypted in any way chosen by the ideal-process adversary. (Since the attacker obtains no information about legitimately-encrypted messages, we are guaranteed that illegitimate ciphertexts will be decrypted to values that are independent from these messages.) Note that the same illegitimate ciphertext can be decrypted to different values in different activations. This provision allows the decryption algorithm to be non-deterministic with respect to ciphertexts that were not legitimately generated.

Another characteristic of $\mathcal{F}_{\text{PKE}}$ is that, when activated with a KeyGen request, it always responds with an (adversarially-chosen) encryption key pk'. Still, only the first key to be generated is recorded, and only messages that are encrypted with that key are guaranteed to remain secret. Messages encrypted with other keys are disclosed to the adversary in full. This modeling represents the fact that a single copy of the functionality captures the security requirements of only a single instance of a public-key encryption scheme (i.e., a single pair of encryption and decryption keys). Other keys may provide correct encryption and decryption, but do not guarantee any security (see [CKN03] for further discussion about possible alternative formulations of the functionality).