Chosen-Ciphertext Secure Proxy Re-Encryption

Ran Canetti* Susan Hohenberger†

May 8, 2007

Abstract

In a proxy re-encryption (PRE) scheme, a proxy is given special information that allows it to translate a ciphertext under one key into a ciphertext of the same message under a different key. The proxy cannot, however, learn anything about the messages encrypted under either key. PRE schemes have many practical applications, including distributed storage, email, and DRM. Previously proposed re-encryption schemes achieved only semantic security[1]; in contrast, applications often require security against chosen ciphertext attacks. We propose a definition of security against chosen ciphertext attacks for PRE schemes, and present a scheme that satisfies the definition. Our construction is efficient and based only on the Decisional Bilinear Diffie-Hellman assumption *in the standard model.* We note that capturing CCA security for PRE schemes turns out to be tricky; we present a game-based definition alongside *two* universally composable definitions, and show that the game-based definition resides between the two.

Keywords: encryption, re-encryption, chosen-ciphertext security, obfuscation

1 Introduction

Encryption is one of the most fundamental cryptographic functions, and yet its practical adoption is often hampered by key management problems. Suppose pk_1 and pk_2 are two independently chosen encryption keys. As pointed out by Mambo and Okamoto [22], it is a common situation in practice that data is encrypted under pk_1 and an application requires that it be encrypted under pk_2. When the holder of sk_1 is online, this translation is easy: $E_1(m)$ is decrypted using sk_1 to obtain m, then m is encrypted under pk_2, resulting in $E_2(m)$. Yet in many applications— encrypted email forwarding [6, 5], distributed file systems [1, 2], and the DRM of Apple's iTunes [25] —this translation is being performed by an untrusted party! As a demonstration of the email forwarding scenario, imagine that Alice is going on vacation and wishes to have her mail server forward all of her encrypted email to Bob, *without giving her secret key to either the mail server or Bob.*

In Eurocrypt 1998, Blaze, Bleumer and Strauss (BBS) proposed a solution to this widely-encountered key management problem [6, 5]. They introduced the concept of *proxy re-encryption*, where a (potentially untrusted) proxy is given a re-encryption key $rk_{1,2}$ that allows it to translate a message m encrypted under public key pk_1 into an encryption of the same message m under a different public key pk_2 – without being able to learn anything about the encrypted messages! I.e., the CPA-security of encryptions under pk_1 and pk_2 hold even against an adversary possessing $rk_{1,2}$.

BBS categorized two types of re-encryption schemes. If the re-encryption key $rk_{1,2}$ necessarily allows the proxy to translate ciphertexts under pk_1 into ciphertexts under pk_2 *and vice versa,*

*IBM T.J. Watson Research Center, Hawthorne, NY, USA, `canetti@watson.ibm.com`

†IBM Research, Zurich Research Laboratory, CH-8803 Rüschlikon, `sus@zurich.ibm.com`

[1]Simultaneously with our work, a CCA-secure scheme in the random oracle model was proposed, as we discuss herein.

then we say that the scheme is *bidirectional.* If the re-encryption key $rk_{1,2}$ allows the proxy to translate only from pk_1 to pk_2, then we say that the scheme is *unidirectional.* (Clearly, any unidirectional scheme can be easily transformed to a bidirectional one. We do not know if the converse holds.) BBS proposed the first bidirectional CPA-secure scheme, leaving the construction of a unidirectional scheme as an open problem. Seven years later, Ateniese et al. [1, 2] presented the first unidirectional CPA-secure scheme. Capturing this elusive unidirectional property came at a price: the re-encryption algorithm was *single-hop*; that is, a re-encrypted ciphertext could not be further re-encrypted. In contrast, the BBS scheme is *multi-hop*, namely a ciphertext can be re-encrypted from Alice to Bob to Carol and so on.

However, both of these PRE algorithms are only CPA-secure, and CPA security is often not sufficient to guarantee security in general protocol settings.

1.1 Our Contributions

We address the problem of obtaining PRE schemes that are secure in arbitrary protocol settings, or in other words are secure against *chosen ciphertext attacks.* The concept of a CCA secure PRE scheme sounds almost self-contradictory, since on the one hand we want the ciphertexts to be non-malleable, and on the other hand we also want to allow the proxy to "translate" the ciphertext from one public key to another. Still, we formulate a meaningful definition of CCA-secure PRE schemes, along with a construction that meets the definition in the standard model and under previously used hardness assumptions. Below we give more details on the construction, the definitions, and the impact of this work.

Constructions. We present the first re-encryption scheme secure against chosen-ciphertext attacks (see the discussion on [17] below). Our scheme is bidirectional and multi-hop (that is, ciphertexts may be re-encrypted from Alice to Bob to Carol and so on).[2] Moreover, our scheme is efficient enough to be used in practice. We prove it secure under the Decisional Bilinear-Diffie Hellman assumption. We first present a scheme and proof in the random oracle model to provide intuition into the techniques we employ in the more complicated proof of the scheme in the standard model.

As in other re-encryption schemes [6, 5, 1, 2, 20, 17], we work in a static adversary model and require that each user choose their key-pairs independently. To model this second requirement, we can either work in the trusted key generation model (where a trusted party generates and distributes all key-pairs) or alternatively, we can work in the registered keys model of Barak et al. [3] (where each user must register by proving knowledge of their secret key).

Definitions. We present a formal definition of security against CCA attacks for re-encryption schemes. The definition gives the adversary access to a re-encryption oracle (which translates ciphertexts) and a re-encryption key oracle (which returns re-encryption keys). For the definition to make sense, one needs to define the game so as to disallow decryption queries not only on the challenge ciphertext (as usual), but also on any trivial derivative of the challenge ciphertext (e.g., derivatives obtained from a re-encryption). We note that demonstrating that the proposed scheme satisfies this definition turns out to be non-trivial; in particular, in some cases, one must correctly answer re-encryption queries *without* knowing the corresponding re-encryption key. In fact, the technique we use in the analysis of the scheme is new and may be useful elsewhere.

To investigate the composability of re-encryption with other protocols, we also formulate a definition within the universally composable (UC) framework [9]. It was not obvious how to extend the ideal functionality for CCA-secure encryption [9] to the case of re-encryption, since additional

[2]The bidirectional scheme of BBS [6, 5] was also multi-hop; all known unidirectional schemes are single-hop (or the size of the ciphertext grows linearly with the number of hops).

key-pairs and oracle queries must be handled. Indeed, we present *two* natural ideal functionalities for re-encryption and show that the power the game-based definition resides between the two.

We note that the proof that the game-based definition implies the (weak) universally composable one uses some similar techniques as the corresponding proof in the case of replayable CCA (RCCA) security [12]. Indeed, the two notions both aim at guaranteeing illegitimate "mauling" of ciphertexts, while permitting legitimate "re-encrypting" without modifying the hidden message.

We leave open many interesting problems in this area, such as building: (1) unidirectional CCA-secure schemes without random oracles, (2) any construction that is simultaneously unidirectional and multi-hop, (3) any unidirectional or CCA-secure scheme without bilinear groups, and (4) secure obfuscations of CCA-secure re-encryption or other key translation schemes.

Practical Impact. In 2005, the digital rights management (DRM) of Apple's iTunes was compromised due to the fact than an untrusted party (i.e., the client's resource) could obtain the plaintext during a naive decrypt-and-encrypt operation, albeit with symmetric encryption [25]. This flaw could have been prevented by using a secure PRE scheme.

In fact that same year, Ateniese et al. [1, 2] built a secure distributed file system, using their re-encryption scheme, where a server can re-encrypt and send files to many different clients without ever gaining access to the plaintext. This also makes the server a less desirable target for hackers.

However, plain CPA-security is clearly not enough for some applications, such as encrypted email forwarding. For instance, an adversary might hope to gain access to a "decryption oracle" by mangling ciphertexts, emailing them to Alice, and then hoping that she responds with, "Did you send the following to me? [Decrypted attachment.]" The present work fills this gap.

Theoretical Impact. Recently, the notion of re-encryption was considered in the more general framework of program obfuscation. Hohenberger, Rothblum, shelat, and Vaikuntanathan [20] presented a unidirectional, CPA-secure re-encryption scheme, where the re-encryption key allows the proxy to learn *nothing* more than black-box access to such a program. This was the first positive result for obfuscating an encryption functionality and stands in high contrast to a series of impossibility results [4] for *general obfuscation* and negative improbability [19, 15] results for obfuscation of many cryptographic functionalities. Here we further relax this notion and only require that the proxy learn nothing about the encrypted messages (but the proxy might learn other things not exposed by black-box access, e.g. it may be able to link between a ciphertext and its re-encryption.) In particular, our re-encryption program is *deterministic*, given the two corresponding public keys; in contrast, the notion of [20] mandates that the re-encryption process be randomized. In other words, our re-encryption definition can be viewed as a meaningful relaxation of program obfuscation for our specific context.

1.2 Intuition Behind the Construction

The idea behind our construction begins with the Canetti, Halevi, and Katz [11] paradigm for transforming any selective-identity, CPA-secure identity-based encryption (IBE) scheme into a CCA-secure cryptosystem. Recall the CHK transformation. Let $(\mathcal{G}, \mathcal{S}, \mathcal{V})$ be a strongly-unforgeable one-time signature scheme. Let (G, E, D) be a semantically-secure IBE scheme. Then a CCA-secure cryptosystem can be created as follows. A user's public key corresponds to the master public key output by G. To encrypt a message m, an encryptor first runs $\mathcal{G}$ to obtain a signature keypair (svk, ssk). The encryptor then generates the ciphertext $c = E(svk, m)$, using svk as the identity, and signs this ciphertext as $s = \mathcal{S}(ssk, c)$. The output of the encryption algorithm is the tuple (svk, c, s). To decrypt, a user first checks that $\mathcal{V}(svk, c, s)$ verifies, and if so, proceeds to decrypt c using the master secret key.

Now, there are many similarities between the Boneh and Franklin IBE [8] and the Ateniese et al. PRE [1]. Can we simply apply the CHK paradigm to achieve CCA security? Unfortunately, applying the CHK paradigm to the Boneh-Franklin IBE looks unwieldy at first. These IBE ciphertexts have two parts: let $c = (X, Y)$. If the encryptor signs (X, Y) in the CHK transformation, then the proxy can't re-encrypt (X, Y) without invalidating the signature. But if the encryptor only signs, say, Y, then the adversary can arbitrarily mutate X, thus changing the decryption value. Our solution is to add an element Z to the ciphertext, such that (Y, Z) will be signed and Z allows anyone to check that the unsigned value X wasn't mutated in any meaningful way.

The primary technical challenge is to prove this scheme secure via a reduction that can successfully answer the numerous oracle queries allowed to the re-encryption adversary. We first show how to navigate these queries in Section 4.1 in the random oracle model, which simplifies the analysis. We then show in Section 4.2 how to replace the random oracles with *specific* concrete hash functions, which will be carefully manipulated to (among other things) sometimes allow the reduction to decrypt or re-encrypt a ciphertext without knowing the correct keys. Our final scheme is only slightly less efficient than its random oracle counterpart.

1.3 Related Work

There is a large related body of work with the very similar name *proxy encryption* (no "re-") [21, 26, 13]. In proxy encryption, Alice allows Bob to decrypt ciphertexts meant for her with the help of a proxy. In these schemes, Alice's secret key is shared between Bob and the proxy. An encryption for Alice is first partially decrypted by the proxy, and then fully decrypted by Bob. However, this approach requires that Bob obtain and store an additional secret (for every decryption delegation he accepts!). Proxy encryption schemes are currently realized under a broader class of complexity assumptions than PREs. For example, Dodis and Ivan present both CPA and CCA secure constructions based on RSA, Decisional DH, and bilinear assumptions [13]. PREs are a (strict) subset of proxy encryption schemes [1], where Bob need only have its own decryption key.

In a concurrent and independent work, Green and Ateniese presented the first CPA and CCA-secure *identity-based* PREs in the random oracle model [17]. They employ the above key sharing technique, but alter it as follows: Alice gives one share of her key to the proxy in the clear and the other share to the proxy encrypted under Bob's public key. To perform re-encryption, the proxy first partially decrypts the ciphertext and then attaches the ciphertext containing the secret share so that Bob can decrypt the rest. In the CCA case, this requires carefully ensuring that the composition of these ciphertexts results in a non-malleable joint-ciphertext. Their CCA IBE scheme is unidirectional and single-hop in the random oracle model, whereas ours is bidirectional and multi-hop in the standard model. Depending on the application, one may need one or the other. One additional difference between the schemes is that our original and re-encrypted ciphertexts come from the same distribution and thus when Bob receives a re-encrypted ciphertext he cannot tell (based on the ciphertext) whether it is an original or re-encrypted one. In [17], original and re-encrypted ciphertexts come from two separate distributions and moreover Bob *must* know who the ciphertext was originally sent to in order to decrypt a re-encryption. Green and Ateniese [17] also present a game-based CCA IBE security definition, which shares many properties with ours, but we relax our definition (in the style of RCCA) and explore the proper UC formulations for re-encryption.

This work should not be confused with the "universal re-encryption" literature [16], which *re-randomizes* ciphertexts instead of changing the *public key* under which they are encrypted.

2 Definitions

Throughout this section we concentrate on defining *bidirectional* re-encryption schemes. The case of *unidirectional* schemes, namely where a re-encryption key from pk_1 to pk_2 should *not* provide the ability to re-encrypt from pk_2 to pk_1 can be inferred in a straightforward way. We allow re-encryption proxies between uncorrupted parties to be corrupted adaptively during the course of the computation. In contrast, we restrict the adversary to determine the identities of the corrupted key-holders before the computation starts. Furthermore, do not allow adaptive corruption of proxies between corrupted and uncorrupted parties.[3]

We first give the input-output specifications for a proxy re-encryption scheme (PRE). Section 2.1 gives a game-based security definition. Section 2.2 gives a security definition within the Universally Composable security framework. Section 2.3 discusses the relationship between the definitions.

Definition 2.1 (Bidirectional PRE Input/Output) *A* bidirectional, proxy re-encryption scheme PRE *is a tuple of PPT algorithms* (KeyGen, ReKeyGen, Enc*,* ReEnc, Dec)*:*

- $\mathsf{KeyGen}(1^k) \rightarrow (pk, sk)$. *On input the security parameter 1^k, the key generation algorithm* KeyGen *outputs a public key pk and a secret key sk.*
- $\mathsf{ReKeyGen}(sk_1, sk_2) \rightarrow rk_{1\leftrightarrow 2}$. *On input two secret keys sk_1 and sk_2, the re-encryption key generation algorithm* ReKeyGen *outputs a bidirectional re-encryption key $rk_{1\leftrightarrow 2}$.*
- $\mathsf{Enc}(pk, m) \rightarrow C$. *On input a public key pk and a message $m \in \{0,1\}^*$, the encryption algorithm* Enc *outputs a ciphertext C.*
- $\mathsf{ReEnc}(rk_{1\leftrightarrow 2}, C_1) \rightarrow C_2$. *On input a re-encryption key $rk_{1\leftrightarrow 2}$ and a ciphertext C_1, the re-encryption algorithm* ReEnc *outputs a second ciphertext C_2 or the error symbol $\perp$.*
- $\mathsf{Dec}(sk, C) \rightarrow m$. *On input a secret key sk and a ciphertext C, the decryption algorithm* Dec *outputs a message $m \in \{0,1\}^*$ or the error symbol $\perp$.*

Remark 2.2 (On generating re-encryption keys:) Here the re-encryption key generation is treated as an algorithm that takes for input the two relevant secret keys and generates a re-encryption key. Alternatively, a re-encryption key can be generated via a protocol involving the proxy and the two holders of the secret keys. The requirements are that the proxy learns the re-encryption key as defined here and nothing else. The holders of the secret keys learn nothing from the protocol. These security requirements must hold only when at most *one* of the three parties is corrupted. When any *two* of the parties are corrupted, we no longer have security requirements.

The input/output specification for a *unidirectional* PRE scheme would change as follows: instead of taking two secret keys as input, the ReKeyGen algorithm would take as input a secret key sk_1 and a public key pk_2, and output a re-encryption key $rk_{1\rightarrow 2}$. Here the protocol for providing a proxy with a re-encryption key form sk_1 to sk_2 is simple: The owner of sk_1 locally computes $rk_{1\rightarrow 2}$ and sends it to the proxy. The owner of sk_2 need not be part of this process.

2.1 Defining Security: A Game-Based Approach

We start by formulating the requirements for correctness of decryption.

Definition 2.3 (Bidirectional PRE Correctness) *A bidirectional* PRE *scheme* (KeyGen*,* ReKeyGen*,* Enc*,* ReEnc*,* Dec) *is* perfectly correct *with respect to domain D if:*

- *For all (pk, sk) output by* KeyGen *and all $m \in D$, it holds that* $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m$*;*
- *For any $n > 1$, any sequence of pairs (pk_1, sk_1),...,$(pk_n,\ sk_n)$ output by* KeyGen*, any $i < n$, all re-encryption keys $rk_{i\leftrightarrow i+1}$ output by* $\mathsf{ReKeyGen}(sk_i, sk_{i+1})$*, all messages $m \in D$ and all C_1 output by* $\mathsf{Enc}(pk_1, m)$*, it holds that* $\mathsf{Dec}(sk_n, \mathsf{ReEnc}(rk_{n-1\leftrightarrow n}, ...\mathsf{ReEnc}(rk_{1\leftrightarrow 2}, C_1)...) = m$*.*
If for any $m \in D$ correctness holds only with probability 1 minus a negligible quantity, we say that the scheme is correct *with respect to D.*

[3]This limitation reflects the current state of the art: Unfortunately, we do not have notions of security that adequately capture adaptive corruption of principals, and are at the same time realizable by realistic protocols.

Next we define the game used for formulating the security requirement. The game defines an interaction between an adversary and a number of oracles, representing the capabilities of the adversary in an interaction with a PRE scheme. It proceeds as follows:

Definition 2.4 (Bidirectional PRE-CCA game) *Let k be the security parameter. Let $\mathcal{A}$ be an oracle TM, representing the adversary. The game consists of an execution of $\mathcal{A}$ with the following oracles, which can be invoked multiple times in any order:*

Uncorrupted key generation: *Obtain a new key pair as $(pk, sk) \leftarrow \mathsf{KeyGen}(1^k)$. $\mathcal{A}$ is given pk.*

Corrupted key generation: *Obtain a new key pair as $(pk,\ sk) \leftarrow \mathsf{KeyGen}(1^k)$. $\mathcal{A}$ is given pk, sk.*[4]

Re-encryption key generation $\mathcal{O}_{rkey}$: *On input (pk, pk') by the adversary, where pk, pk' were generated before by* KeyGen*, return the re-encryption key $rk_{pk \leftrightarrow pk'} = \mathsf{ReKeyGen}(sk, sk')$ where sk, sk' are the secret keys that correspond to pk, pk'. Here we require that either both pk and pk' are corrupted, or alternatively both are uncorrupted. We do not allow for re-encryption key generation queries between a corrupted an uncorrupted key. (This represents the restriction that the identities of parties whose security is compromised should be fixed in advance.)*

Challenge oracle: *This oracle can be queried only once. On input (pk^*, m_0, m_1), where pk^* is called the* challenge key*, the oracle chooses a bit $b \leftarrow \{0, 1\}$ and returns the* challenge ciphertext *$C^* = \mathsf{Enc}(pk^*, m_b)$. (As we note later, the challenge key must be uncorrupted for $\mathcal{A}$ to win.)*

Re-encryption $\mathcal{O}_{renc}$: *On input (pk, pk', C), where pk, pk' were generated before by* KeyGen*, if pk' is corrupted and (pk, C) is a* derivative *of (pk^*, C^*), then return a special symbol $\perp$ which is not in the domains of messages or ciphertexts. Else, return the re-encrypted ciphertext $C' = \mathsf{ReEnc}(\mathsf{ReKeyGen}(sk,\ sk'), C)$. Derivatives of (pk^*, C^*) are defined inductively, as follows. (See informal discussion immediately below.)*

1. *(pk^*, C^*) is a derivative of itself.*
2. *If (pk, C) is a derivative of (pk^*, C^*) and (pk', C') is a derivative of (pk, C) then (pk', C') is a derivative of (pk^*, C^*).*
3. *If $\mathcal{A}$ has queried the re-encryption oracle $\mathcal{O}_{renc}$ on input (pk, pk', C) and obtained response (pk', C'), then (pk', C') is a derivative of (pk, C).*
4. *If $\mathcal{A}$ has queried the re-encryption key generation oracle $\mathcal{O}_{rkey}$ on input (pk, pk') or (pk', pk), and $\mathsf{Dec}(pk', C') \in \{m_0, m_1\}$, then (pk', C') is a derivative of (pk, C).*

Decryption oracle $\mathcal{O}_{dec}$: *On input (pk, C), if the pair $(pk,\ C)$ is a derivative of the challenge ciphertext C^*, or pk was not generated before by* KeyGen*, then return a special symbol $\perp$ which is not in the domain D of messages. Else, return $\mathsf{Dec}(sk, C)$.*

Decision oracle: *This oracle can also be queried only once. On input b': If $b' = b$ and the challenge key pk^* is not corrupted, then output 1; else output 0.*

We say that $\mathcal{A}$ wins the PRE-CCA game with advantage *ε if the probability, over the random choices of $\mathcal{A}$ and the oracles, that the decision oracle is invoked and outputs 1, is at least $1/2 + \varepsilon$.*

We motivate the above definition of derivatives, which is at the heart of the notion of security. Informally, a pair (pk, C) is a derivative of (pk^*, C^*) if decrypting C with the secret key sk that corresponds to pk would give the adversary "illegitimate information" on the hidden bit b. The first three conditions are obvious. The fourth condition represents the fact that if the adversary has the re-encryption key between pk^* and pk (or alternatively a chain of re-encryption keys $(pk^*, pk'''), (pk''', pk''), ..., (pk', pk)$) then it is possible that C is the result of legitimately re-encrypting C^* to key pk, in which case decrypting C would give the adversary "illegitimate" information on b.

[4]Alternatively, the adversary $\mathcal{A}$ could register her public key with an authority by proving knowledge of the corresponding secret key, as in the registered keys model of Barak et al. [3].

A first attempt to prevent this may be to not allow the adversary to decrypt *any* ciphertext with respect to key pk. However, such a rule would be too restrictive (resulting in an overly weak definition), since C might be generated *not* as a re-encryption of C^*, in which case we want to allow the adversary to obtain the decryption of C. Furthermore, telling whether C was generated as a re-encryption of C^* may be problematic, especially when the re-encryption algorithm is randomized. A second attempt may be to require that the C is not the result of running the re-encryption algorithm on the challenge ciphertext C^*, for *any* randomness. However, this too would result in an overly weak definition, since a re-encryption algorithm could artificially output *any* string in $\{0,1\}^*$ with negligible but positive probability. Restricting to non-negligible outputs does not seem to suffice either.

We circumvent this difficulty by allowing the adversary to decrypt C, while making sure that if C was generated as a re-encryption of C^* then the adversary gains no information from the decryption query. This definitional approach is reminiscent of the definition of *Replayable CCA security* in [12]. In both cases we want to guarantee "CCA security" while allowing re-encryptions that do not change the decrypted value.

We note that this definition too results in a relaxation of plain CCA security, in the same way that RCCA relaxes CCA secrurity. That is, the present definition allows "harmless mauling" of the ciphertext to a different ciphertext that decrypts to the same value (see discussion in [12].) We view this relaxation as an additional feature of the present definition. In particular, it allows for potential anonymization of the re-encryption process along the lines of [12, 18, 23]. Such an anonymization is ruled out by strict CCA security. The relations with the UC notions of security provide aditional justification for the adequacy of the definition.

Definition 2.5 (Bidirectional PRE-CCA security) *A PRE scheme is Bidirectional PRE-CCA secure for domain D of messages if it is correct for D as in Definition 2.3, and any PPT adversary wins the bidirectional PRE-CCA game only with negligible advantage.*

Remark 2.6 (On Private Re-Encryption Keys.) *We point out that this captures the important feature that, from a ciphertext and its re-encrypted value, the adversary does not learn the re-encryption keys. Suppose the adversary could learn a re-encryption key $rk_{c\leftrightarrow d}$ from queries to $\mathcal{O}_{renc}$, then it could always win the PRE game by re-encrypting challenge (c, C_b) to the pair (d, D_b) using $rk_{c\leftrightarrow d}$, and then querying the decryption oracle $\mathcal{O}_{dec}$ on (d, D_b).*

2.2 UC Definitions of Secure Proxy Re-Encryption

We also formulate a definition for proxy re-encryption within the universally composable (UC) security framework, and investigate its relation with the game-based definition. In addition to providing additional confidence in the adequacy of the notion, a definition in the UC framework carries with it strong composability guarantees.

Providing a UC definition amounts to formulating an *ideal functionality* for proxy re-encryption. However, precisely capturing the security guarantees provided by Definition 2.5 as an ideal functionality within the UC framework turns out to be non-trivial; see more discussion in Appendix A. We thus provide *two* such definitions, namely two formulations of an "ideal proxy re-encryption functionality", and show that one formulation implies Definition 2.5, whereas the second formulation is implied by Definition 2.5. This allows us to "sandwich" the security guarantees provided by Definition 2.5, when formulated in a composable, simulation based framework.

In a nutshell, we consider a weak functionality, $\mathcal{F}_{\texttt{wPRE}}$, where the adversary is not allowed to make re-encryption queries from uncorrupted to corrupted parties. We then consider a stronger functionality, $\mathcal{F}_{\texttt{sPRE}}$, where the adversary is allowed to make re-encryption queries between any two parties (as she is in Definition 2.5.) UC security is defined as follows. Details in Appendix A.

Definition 2.7 (UC-secure PRE schemes) *A* PRE *scheme* Σ *is non-adaptively strongly (resp., weakly) UC-secure if the protocol* π_Σ *UC-realizes* $\mathcal{F}_{\texttt{sPRE}}$ *(resp.,* $\mathcal{F}_{\texttt{wPRE}}$*) as defined in [9].*

2.3 Relationships between Security Definitions

The following relationships between these definitions hold. Proof of Theorem 2.9 is in Appendices B and C.

Theorem 2.8 (Strongly UC implies Game-Based) *Let* Σ *be a* PRE *scheme that's non-adaptively* **strongly** *UC-secure as in Definition 2.7. Then* Σ *is bidirectional* PRE-*CCA re-encryption scheme (as in Definition 2.5).*

Theorem 2.9 (Game-Based implies Weakly UC) *Let* Σ *be a bidirectional* PRE-*CCA re-encryption scheme (as in Definition 2.5) over message space* D*, where* $|D|$ *is super-polynomial in the security parameter.*[5] *Then it holds that* Σ *is non-adaptively* **weakly** *UC-secure as in Definition 2.7.*

3 Preliminaries

Bilinear Groups: We write $\mathbb{G} = \langle g \rangle$ to denote that g generates the group $\mathbb{G}$. Let BSetup be an algorithm that, on input the security parameter 1^k, outputs the parameters for a bilinear map as $(q, g, h, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, where $\mathbb{G}, \mathbb{G}_T$ are of prime order $q \in \Theta(2^k)$ and $\langle g \rangle = \langle h \rangle = \mathbb{G}$. The efficient mapping[6] $\mathbf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is both: (*Bilinear*) for all $g \in \mathbb{G}$ and $a, b \in \mathbb{Z}_q^2$, $\mathbf{e}(g^a, g^b) = \mathbf{e}(g,g)^{ab}$; and (*Non-degenerate*) if g generates $\mathbb{G}$, then $\mathbf{e}(g,g) \neq 1$.

The security of our schemes depend only on this mild assumption:

Decisional Bilinear Diffie-Hellman (DBDH) [8]: Let $\mathsf{BSetup}(1^k) \to (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, where $\langle g \rangle = \mathbb{G}$. For all p.p.t. adversaries $\mathcal{A}$, the following probability is strictly less than $1/2 + 1/\text{poly}(k)$:

$$\Pr[a, b, c, d \leftarrow \mathbb{Z}_q;\ x_0 \leftarrow \mathbf{e}(g,g)^{abc};\ x_1 \leftarrow \mathbf{e}(g,g)^d;\ z \leftarrow \{0,1\}; z' \leftarrow \mathcal{A}(g, g^a, g^b, g^c, x_b) : z = z'].$$

It will simplify the reading of our proofs to use the following *equivalent* formulation of DBDH, known as **Modified DBDH** [24]. The modified DBDH assumption is identical to the DBDH assumption, except that $x_0 \leftarrow \mathbf{e}(g,g)^{ab/c}$ (instead of $x_0 \leftarrow \mathbf{e}(g,g)^{abc}$).

Lemma 3.1 *mDBDH holds in* $(\mathbb{G}, \mathbb{G}_T)$ *if and only if DBDH holds in* $(\mathbb{G}, \mathbb{G}_T)$*. Moreover, if mDBDH is solvable with probability* ε*, then DBDH is solvable with probability* ε*; and vice versa.*

Proof. (DBDH $\Longrightarrow$ mDBDH.) On DBDH input (g, g^a, g^b, g^c, Q), query the mDBDH solver on input $(g^c, g^a, g^b, g, Q) = (y, y^A, y^B, y^C, Q)$ and output its response. Observe that mDBDH will output 1 if and only if $Q = \mathbf{e}(y,y)^{ABC}$, which by substitution yields, $\mathbf{e}(g^c, g^c)^{(a/c)(b/c)(1/c)} = \mathbf{e}(g,g)^{ab/c}$.
(mDBDH $\Longrightarrow$ DBDH.) Omitted. □

4 CCA-Secure, Bidirectional Re-Encryption Constructions

We first present a simple construction in the random oracle model, and then show how replace the random oracles with concrete hash functions.

[5] When D is small, our formulation of $\mathcal{F}_{\texttt{wPRE}}$ in Appendix A becomes unnecessarily restrictive. This restrictiveness can be overcome at the price of a more cumbersome formulation. Our schemes have large D.

[6] As we've written our bilinear map above, we've restricted ourselves to certain implementations, e.g., supersingular curves. This is done for clarity of exposition. We could implement our construction using a more general bilinear mapping of the form $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where efficient isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ may not exist. A benefit of using more general curves is that we could reduce the bits per ciphertext transmitted (see [14] for more details).

4.1 PRE Construction, Π_{RO}, in the Random Oracle Model

Notation and Configuration. Let 1^k be the security parameter and $(q, g, h, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ be the bilinear map parameters output by $\mathsf{BSetup}(1^k)$. Let $\mathsf{Sig} = (\mathcal{G}, \mathcal{S}, \mathcal{V})$ be a strongly unforgeable one-time signature scheme, where $\ell = \ell_{sig}(k)$ denotes the length of the verification keys output by $\mathcal{G}(1^k)$. Moreover, we assume that the verification key space produced by $\mathcal{G}$ has super-logarithmic minimum entropy; that is, any given key has a negligible chance of being sampled. Let $H : \{0,1\}^{\leq \ell} \to \mathbb{G}$ and $F : \{0,1\}^{\leq \ell} \to \mathbb{G}$ be two independent hash functions, which we will treat as random oracles.

Define the algorithm *Check* on input a ciphertext tuple (A, B, C, D, E, S) and a key pk as follows:

1. Run $\mathcal{V}(A, (C, D, E), S)$ to verify signature S on message (C, D, E) with respect to key A.
2. Check that $\mathbf{e}(B, F(A)) = \mathbf{e}(pk, D)$ and that $\mathbf{e}(B, h) = \mathbf{e}(pk, E)$.
3. If any of these checks fail, output 0; else output 1.

Scheme $\Pi_{RO} = (\mathsf{KeyGen}, \mathsf{ReKeyGen}, \mathsf{Enc}, \mathsf{ReEnc}, \mathsf{Dec})$ is described as follows:

Key Generation (KeyGen): On input 1^k, select random $x \in \mathbb{Z}_q$. Set $pk = g^x$ and $sk = x$.

Re-Encryption Key Generation ($\mathsf{ReKeyGen}$): On input $sk_X = x$ and $sk_Y = y$, output the bidirectional re-encryption key $rk_{X \leftrightarrow Y} = x/y \mod q$.

Encryption (Enc): On input pk and a message $m \in \mathbb{G}_T$, do:

1. Select a one-time signature keypair as $\mathcal{G}(1^k) \to (svk, ssk)$. Set $A = svk$.
2. Select a random $r \in \mathbb{Z}_q$ and compute
$$B = pk^r \quad , \quad C = \mathbf{e}(g, H(svk))^r \cdot m,$$
$$D = F(svk)^r \quad , \quad E = h^r.$$
3. Run the signing algorithm $\mathcal{S}(ssk, (C, D, E))$, where the message to sign is the tuple (C, D, E), and denote the signature S.
4. Output the ciphertext (A, B, C, D, E, S).

Re-Encryption (ReEnc): On input a re-encryption key $rk_{X \leftrightarrow Y} = x/y$ and a ciphertext $K = (A, B, C, D, E, S)$ under key pk_Y, re-encrypt the ciphertext to be under key pk_X as:

1. Compute $B' = B^{rk_{X \leftrightarrow Y}} = g^{(yr)(x/y)} = g^{xr}$.
2. If $Check(K, pk_Y) = 1$, output the new ciphertext (A, B', C, D, E, S); otherwise, output $\perp$.

Decryption (Dec): On input a secret key sk and any ciphertext $K = (A, B, C, D, E, S)$, if $Check(K, g^{sk}) = 1$, then output the message $C/\mathbf{e}(B, H(A))^{1/sk}$; otherwise, output $\perp$.

Remark 4.1 *Parties X and Y can compute $rk_{X \leftrightarrow Y}$ using generic secure function evaluation techniques or, more efficiently, X may select a random $r \in \mathbb{Z}_q$ and send $(rx \mod q)$ to Y and r to the proxy. Y then sends $(rx/y \mod q)$ to the proxy. The proxy computes $(x/y \mod q)$. Recall that in a bidirectional scheme, no security is guaranteed if the proxy colludes with either party.*

Theorem 4.2 *If the DBDH assumption holds in $(\mathbb{G}, \mathbb{G}_T)$, then scheme Π_{RO} is a Bidirectional* $\mathtt{PRE}$-*CCA secure for domain $\mathbb{G}_T$ of messages in the random oracle model.*

Proof sketch. Recall that DBDH and mDBDH are equivalent. (It will be less tedious to use mDBDH here.) We begin by observing that if the mDBDH assumption holds in $(\mathbb{G}, \mathbb{G}_T)$, then there exist strongly unforgeable one-time signature schemes. Let $\mathsf{Sig} = (\mathcal{G}, \mathcal{S}, \mathcal{V})$ be such a scheme, where $\mathcal{G}$

has super-logarithmic minimum entropy. Let $\mathcal{A}$ be any p.p.t. adversary. Then, we show how to construct a p.p.t. adversary $\mathcal{B}$, with black-box access to $\mathcal{A}$, that succeeds in breaking the mDBDH assumption with probability:

$$\Pr[\mathcal{B} \text{ breaks mDBDH}] \geq 1/2 + \Pr[\mathcal{A} \text{ breaks } \Pi]/2 - \Pr[\mathcal{A} \text{ breaks } \mathsf{Sig}] - q_O \cdot \delta,$$

where $\mathcal{A}$ makes q_O oracle queries and δ is the maximum probability that any given verification key is output by $\mathcal{G}$ (which by assumption is negligible). Also by assumption, $\Pr[\mathcal{A} \text{ breaks } \mathsf{Sig}]$ is negligible for any p.p.t. adversary $\mathcal{A}$. Let us now describe how $\mathcal{B}$ operates.

On mDBDH input (g, g^a, g^b, g^c, Q), where $\mathcal{B}$'s goal is to decide if $Q = \mathbf{e}(g,g)^{ab/c}$ or not, $\mathcal{B}$ sets up the global parameters for $\mathcal{A}$ as follows: the description of the groups $\langle g \rangle = \mathbb{G}, \mathbb{G}_T$, their prime order q, and the mapping $\mathbf{e}$, which are implicit in the mDBDH input, will also be used in the re-encryption game. Set $h = g^{cw}$, where $w \in \mathbb{Z}_q$ is chosen randomly. The system parameters are $(q, g, h, \mathbb{G}, \mathbb{G}_T, \mathbf{e}, H, F)$, where H and F are random oracles. The security parameter is $k \geq |q|$.

$\mathcal{B}$ runs $\mathcal{G}(1^k) \rightarrow (svk^*, ssk^*)$, and records these values. Oracle queries from $\mathcal{A}$ are handled as:

- *Key Generation:* $\mathcal{B}$ chooses a random $x_i \in \mathbb{Z}_q$. If user i is *uncorrupted*, then $\mathcal{B}$ outputs $pk_i = (g^c)^{x_i} = g^{cx_i}$. If user i is *corrupted*, then $\mathcal{B}$ outputs $(pk_i, sk_i) = (g^{x_i}, x_i)$.
- *Hash H:* On input m to hash function H, check to see if pair (m, Y) is recorded in database D_H. If not select random $Y \in \mathbb{G}$, record (m, Y) in D_H and do:

$$H(m) = \begin{cases} g^a & \text{if } m = svk^* \\ Y & \text{otherwise, where } (m, Y) \in D_H. \end{cases} \tag{1}$$

- *Hash F:* On input m to hash function F, check to see if pair (m, Z, s) is recorded in database D_F. If not select random $s \in \mathbb{Z}_q$, record (m, g^s, s) in D_F and do:

$$F(m) = \begin{cases} g^c & \text{if } m = svk^* \\ Z & \text{otherwise, where } (m, Z, s) \in D_F. \end{cases} \tag{2}$$

- *Decryption:* On input (i, K) to $\mathcal{O}_{dec}$, if $Check(K, pk_i) = 0$, then the ciphertext is not well-formed, so $\mathcal{B}$ halts and returns $\perp$. Otherwise, $\mathcal{B}$ proceeds as follows.
  If user i is corrupted, then $sk_i = x_i$ and $\mathcal{B}$ returns $\mathsf{Dec}(sk_i, K)$. If user i is uncorrupted, then $\mathcal{B}$ parses K as (A, B, C, D, E, S). Since the ciphertext is well-formed, we know that $B = pk_i^r$ and $D = F(A)^r$ for the same value of $r \in \mathbb{Z}_q$. Finally, $\mathcal{B}$ checks that $A \neq svk^*$ and aborts if this is false. If all checks pass, then $\mathcal{B}$ finds $(A, F(A), s) \in D_F$ and decrypts the ciphertext by computing $C/\mathbf{e}(D^{1/s}, H(A)) = C/\mathbf{e}(F(A)^{r/s}, H(A)) = C/\mathbf{e}(g^r, H(A))$.
- *Re-Encryption Key:* On input (i, j) to $\mathcal{O}_{rkey}$, if one of i and j is uncorrupted and the other is corrupted, then this call is illegal. Otherwise, $\mathcal{B}$ outputs the re-encryption key x_j/x_i.
- *Re-Encryption:* On input (i, j, K) to $\mathcal{O}_{renc}$, if the value $Check(K, pk_i) = 0$, then the ciphertext is not well-formed, so $\mathcal{B}$ halts and returns $\perp$. Otherwise, $\mathcal{B}$ parses $K = (A, B, C, D, E, S)$.
  - If users i and j are *both* corrupted or if they are *both* uncorrupted, $\mathcal{B}$ computes the re-encryption key x_j/x_i and executes $\mathsf{ReEnc}(x_j/x_i, K)$.
  - If user i is corrupted and user j is uncorrupted, then $\mathcal{B}$ computes $E^{x_j/w} = h^{rx_j/w} = g^{cwrx_j/w} = g^{crx_j} = pk_j^r = B'$ (where $B = pk_i^r$) and outputs (A, B', C, D, E, S).
  - If user i is uncorrupted and user j is corrupted, if $A = svk^*$, then $\mathcal{B}$ outputs $\perp$. Otherwise, $\mathcal{B}$ finds $(A, F(A), s) \in D_F$ and computes $D^{x_j/s} = F(A)^{rx_j/s} = g^{rx_j} = pk_j^r = B'$ (where $B = pk_i^r$) and outputs (A, B', C, D, E, S).

- *Challenge:* At some point, $\mathcal{A}$ will output a challenge tuple (i, m_0, m_1), where i is the index of an uncorrupted user. $\mathcal{B}$ responds choosing a random $d \in \{0,1\}$ and setting:

$$\begin{aligned} A &= svk^* \quad , \quad B = (g^b)^{x_i} = pk_i^{b/c}, \\ C &= Q \cdot m_d \quad , \quad D = g^b = F(A)^{b/c}, \\ E &= (g^b)^w = h^{b/c} \quad , \quad S = \mathcal{S}_{ssk^*}(C, D, E). \end{aligned}$$

- *Decision:* At some point, $\mathcal{A}$ will output a guess $d' \in \{0,1\}$. If $d = d'$, then $\mathcal{B}$ outputs 1 (i.e., mDBDH instance), otherwise $\mathcal{B}$ outputs 0 (i.e., not an mDBDH instance).

The setup, keys and hash responses of $\mathcal{B}$ are perfectly distributed according to a real instance of re-encryption scheme Π_{RO}. The decryption and re-encryption queries are also perfect, except that $\mathcal{B}$ cannot always answer them when $A = svk^*$. First, consider that *before* the challenge is given, $\mathcal{A}$ has a $q_O \cdot \delta$ chance of querying either oracle on a ciphertext with $A = svk^*$. *After* the challenge is given, $\mathcal{A}$'s chance of querying these oracles on a well-formed ciphertext where $A = svk^*$ and yet the ciphertext is not a derivative of the challenge is $\Pr[\mathcal{A} \text{ breaks } \mathsf{Sig}]$. Consider that a *well-formed* ciphertext (A, B, C, D, E, S) decrypts uniquely *regardless* of the corresponding public key. That is, $C = \mathbf{e}(g, H(A))^r \cdot m$ and $D = F(A)^r$ uniquely fixes m. If we recall the definition of a derivative from Definition 2.4, then we see by inspection that if $A = svk^*$ then for the ciphertext not to be a derivative of the challenge (C, D, E) must not be identical to the challenge ciphertext. If the ciphertext is well-formed, then S is a valid forgery against Sig.

When $\mathcal{B}$ receives an mDBDH instance as input, its challenge ciphertext is also a perfectly distributed, proper encryption of message m_d. Thus, in this case, $\mathcal{A}$ guesses $d' = d$ with the same advantage as it would in a real execution of Π_{RO}. To see this, recall that $H(svk^*) = g^a$, and in this case, $Q = \mathbf{e}(g,g)^{ab/c}$. Thus, $C/\mathbf{e}(B, H(A))^{1/sk_i} = \mathbf{e}(g,g)^{ab/c} \cdot m_d / \mathbf{e}(g^{bx_i}, g^a)^{1/cx_i} = m_d$.

When $\mathcal{B}$ does not receive an mDBDH instance, then the challenge ciphertext contains no information about m_d, since ciphertext component C is uniformly distributed in $\mathbb{G}_T$ independent of d, and $\mathcal{A}$ succeeds in predicting $d' = d$ with exactly 1/2 probability.

Finally, recall from Lemma 3.1 that if there exists a p.p.t. adversary that breaks mDBDH with probability ε, then there also exists a p.p.t. adversary that breaks DBDH with probability ε. □

The following is immediate from Theorems 2.9 and 4.4.

Corollary 4.3 *If the DBDH assumption holds in $(\mathbb{G}, \mathbb{G}_T)$, then bidirectional re-encryption scheme Π_{RO} is non-adaptively, weakly UC-secure with respect to Definition 2.7 in the random oracle model.*

4.2 PRE Construction, Π, without Random Oracles

Now, we will remove the random oracles from the construction in Section 4.1. To do so, we will change the way hash functions H and F operate. Our setup is similar to that of Π_{RO}, except that:

Function H: We replace random oracle H with a universal one-way hash function family (following Canetti, Halevi, and Katz [10] and Boneh and Boyen [7].) Let $\mathcal{H}$ be a pairwise independent family of hash functions $H : \{0,1\}^{\leq \ell} \to \mathbb{G}$, where given an element $x \in \{0,1\}^{\leq \ell}$ and $y \in \mathbb{G}$, there is an efficient algorithm for sampling $H \in \mathcal{H}$ such that $H(x) = y$.

Function F: While H can be replaced by *any* universal one-way hash satisfying the above constraints, F will be replaced by the following specific one [10]. Let g_2, g_3 be random generators of $\mathbb{G}$. Then we define the function $F : \mathbb{Z}_q \to \mathbb{G}$ as $F(y) \overset{\text{def}}{=} g_2^{\tilde{y}} \cdot g_3$, where $\tilde{y}$ is a fixed one-to-one representation of y in $\mathbb{Z}_q$. (An additional hash function can be included in the public key to implement this one-to-one mapping from y to $\tilde{y}$.) For simplicity, we will write y instead of $\tilde{y}$.

To change from scheme Π_{RO} to scheme Π only the encryption algorithm changes as:

Encryption (Enc): On input pk and message $m \in \mathbb{G}_T$, do:

1. Select a one-time signature keypair as $\mathcal{G}(1^k) \rightarrow (svk, ssk)$. Set $A = svk$.
2. Select a random $r \in \mathbb{Z}_q$ and compute
$$B = pk^r \quad , \quad C = \mathbf{e}(g, H(svk))^r \cdot m,$$
$$D = F(svk)^r = (g_2^{svk} \cdot g_3)^r \quad , \quad E = h^r.$$
3. Run the signing algorithm $\mathcal{S}(ssk, (C, D, E))$, where the message to sign is the tuple (C, D, E), and denote the signature S.
4. Output the ciphertext (A, B, C, D, E, S).

Efficiency comparison of Π and Π_{RO}. Scheme Π remains surprisingly efficient compared to Π_{RO}. It requires two additional elements from $\mathbb{G}$ in the global parameters. The cost to compute a ciphertext also increases by two multi-base exponentiations in $\mathbb{G}$ to cover the hashes.

We are now ready to present our main result. Removing oracle H is rather straight-forward, because H was only programmed on one point in the proof of Theorem 4.2. Canetti, Halevi, and Katz [10] present one method of designing a universal one-way hash function that satisfies a series of polynomial constraints, i.e., pairs (x, y) such that $H(x) = y$. Removing oracle F, however, is more involved, because F was programmed for *exponentially* many points. In particular, the proof of Theorem 4.2 required that: (1) F satisfied one (x, y) constraint and (2) that for all other inputs $z \neq x$, the discrete log of $F(z)$ base g was known to the party $\mathcal{B}$ playing the security game with the adversary. In our new scheme Π, we specially designed a new function F such that: (1) F satisfies one constraint (x, y) and (2) such that *together with the other parts of the ciphertext*, $\mathcal{B}$ can compute the necessary information related to $F(z)$ base g for all inputs $z \neq x$.

Theorem 4.4 *If the DBDH assumption holds in $(\mathbb{G}, \mathbb{G}_T)$, then scheme Π is a Bidirectional* `PRE`-*CCA secure for domain $\mathbb{G}_T$ of messages in the standard model.*

Proof sketch. We now describe how to modify the proof of Theorem 4.2 to our new scheme without random oracles by specifying a different algorithm $\mathcal{B}$, with black-box access to $\mathcal{A}$, that succeeds in breaking the mDBDH assumption with probability:

$$\Pr[\mathcal{B} \text{ breaks mDBDH}] \geq 1/2 + \Pr[\mathcal{A} \text{ breaks } \Pi]/2 - \Pr[\mathcal{A} \text{ breaks } \mathsf{Sig}] - q_O \cdot \delta,$$

where $\mathcal{A}$ makes q_O oracle queries and δ is the maximum probability that any given verification key is output by $\mathcal{G}$ (which by assumption is negligible.)

On mDBDH input (g, g^a, g^b, g^c, Q), where $\mathcal{B}$'s goal is to decide if $Q = \mathbf{e}(g, g)^{ab/c}$ or not, $\mathcal{B}$ sets up the global parameters for $\mathcal{A}$ as follows: $\mathcal{B}$ runs $\mathcal{G}(1^k) \rightarrow (svk^*, ssk^*)$. Next, $\mathcal{B}$ sets the generators $h := g^{cw}$, $g_2 := g^{\alpha_1}$, and $g_3 := g^{-\alpha_1 svk^*} \cdot g^{c\alpha_2}$ for a randomly chosen $w, \alpha_1, \alpha_2 \in \mathbb{Z}_q^3$. Finally, $\mathcal{B}$ samples a pairwise independent hash function $H \in \mathcal{H}$ such that $H(svk^*) := g^a$. The system parameters are $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e}, h, g_2, g_3, H)$. $\mathcal{B}$ runs $\mathcal{G}(1^k) \rightarrow (svk^*, ssk^*)$, and remembers these values. Oracle queries from $\mathcal{A}$ to $\mathcal{O}_{rkey}$ are handled the same as in the proof of Theorem 4.2. The other oracles change as follows:

- *Key Generation:* $\mathcal{B}$ chooses a random $x_i \in \mathbb{Z}_q$. If user i is *uncorrupted*, then $\mathcal{B}$ outputs $pk_i = (g^c)^{x_i} = g^{cx_i}$. Otherwise, $\mathcal{B}$ sets $sk_i = x_i$, $pk_i = g^{x_i}$, and outputs (pk_i, sk_i).

- *Decryption:* On input (i, K) to $\mathcal{O}_{dec}$, if $Check(K, pk_i) = 0$, then the ciphertext is not well-formed, so $\mathcal{B}$ halts and returns $\perp$. Next, $\mathcal{B}$ checks that $A \neq svk^*$ and aborts if this is false. Otherwise, $\mathcal{B}$ proceeds as follows. Notice that in a well-formed ciphertext, $B = pk_i^r$ and $D = F(A)^r$ for the same value of $r \in \mathbb{Z}_q$. $\mathcal{B}$ decrypts the ciphertext by computing
$$t = \frac{D}{B^{\frac{\alpha_2}{x_i}}} \quad , \quad \lambda = \frac{1}{\alpha_1(A - svk^*)}.$$
Then, $\mathcal{B}$ outputs the message $C/\mathbf{e}(t^\lambda, H(A))$.
Note that when $A \neq svk^*$, then $\mathcal{B}$ can solve for $t^\lambda = g^r$ because:
$$t \;=\; \frac{F(A)^r}{(pk_i^r)^{\frac{\alpha_2}{x_i}}} \;=\; \frac{g_2^{rA} g_3^r}{pk_i^{\frac{r\alpha_2}{x_i}}} \;=\; \frac{(g^{\alpha_1})^{rA}(g^{-\alpha_1 svk^* + c\alpha_2})^r}{(g^{cx_i})^{\frac{r\alpha_2}{x_i}}} \;=\; \frac{g^{r\alpha_1(A - svk^*) + rc\alpha_2}}{g^{rc\alpha_2}} \;=\; g^{r\alpha_1(A-svk^*)}.$$
- *Re-Encryption:* On input (i, j, K) to $\mathcal{O}_{renc}$, if the value $Check(K, pk_i) = 0$, then the ciphertext is not well-formed, so $\mathcal{B}$ halts and returns $\perp$. Otherwise, $\mathcal{B}$ parses $K = (A, B, C, D, E, S)$.
  - If both users i and j are uncorrupted or both are uncorrupted, $\mathcal{B}$ computes the re-encryption key x_j/x_i and executes the algorithm $\mathsf{ReEnc}(x_j/x_i, K)$.
  - If user i is corrupted and user j is uncorrupted, then $\mathcal{B}$ computes $E^{x_j/w} = h^{rx_j/w} = g^{cwrx_j/w} = g^{crx_j} = pk_j^r = B'$ (where $B = pk_i^r$) and outputs (A, B', C, D, E, S).
  - If user i is uncorrupted and user j is corrupted, if $A = svk^*$, then $\mathcal{B}$ outputs $\perp$. Otherwise, $\mathcal{B}$ solves for g^r as it does in decryption (where $B = pk_i^r$), computes $g^{rx_j} = pk_j^r = B'$ and outputs (A, B', C, D, E, S).
- *Challenge:* At some point, $\mathcal{A}$ will output a challenge tuple (i, m_0, m_1), where i is the index of an honest user. $\mathcal{B}$ responds choosing a random $d \in \{0, 1\}$ and setting:
$$\begin{aligned} A = svk^* \quad &, \quad B = (g^b)^{x_i} = pk_i^{b/c}, \\ C = Q \cdot m_d \quad &, \quad D = (g^b)^{\alpha_2} = (g_2^{svk^*} \cdot g_3)^{b/c}, \\ E = (g^b)^w = h^{b/c} \quad &, \quad S = \mathcal{S}_{ssk^*}(C, D, E). \end{aligned}$$
- *Decision:* At some point, $\mathcal{A}$ will output a guess $d' \in \{0, 1\}$. If $d = d'$, then $\mathcal{B}$ outputs 1 (i.e., mDBDH instance), otherwise $\mathcal{B}$ outputs 0 (i.e., not an mDBDH instance).

This ends our description of $\mathcal{B}$. The analysis follows the previous proof. □

The following is immediate from Theorems 2.9 and 4.4.

Corollary 4.5 *If the DBDH assumption holds in $(\mathbb{G}, \mathbb{G}_T)$, then bidirectional re-encryption scheme* Π *is non-adaptively, weakly UC-secure with respect to Definition 2.7 in the standard model.*

References

[1] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. In *NDSS*, pages 29–43, 2005.

[2] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):1–30, February 2006.

[3] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS '04*, pages 186–195, 2004.

[4] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO '01*, volume 2139 of LNCS, pages 1–18, 2001.

[5] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT '98*, volume 1403 of LNCS, pages 127–144, 1998.

[6] Matt Blaze and Martin Strauss. Atomic proxy cryptography. Technical report, AT&T Research, 1997.

[7] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT '04*, volume 3027 of LNCS, pages 223–238, 2004.

[8] Dan Boneh and Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO '01*, volume 2139 of LNCS, pages 213–229, 2001.

[9] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS '01*, 2001. See updated version at Cryptology ePrint Archive: Report 2000/067.

[10] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT '03*, volume 2656 of LNCS, pages 255–271, 2003.

[11] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT '04*, volume 3027 of LNCS, pages 207–222, 2004.

[12] Ran Canetti, Hugo Krawczyk, and Jesper B. Nielsen. Relaxing chosen-ciphertext security. In *CRYPTO '03*, volume 2729 of LNCS, pages 565–582, 2003.

[13] Yevgeniy Dodis and Anca-Andreea Ivan. Proxy cryptography revisited. In *NDSS '03*, 2003.

[14] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers, 2006. Cryptology ePrint Archive: Report 2006/165.

[15] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS '05*, pages 553–562, 2005.

[16] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul F. Syverson. Universal re-encryption for mixnets. In *CT-RSA '04*, volume 2964 of LNCS, pages 163–178, 2004.

[17] Matthew Green and Giuseppe Ateniese. Identity-based proxy re-encryption. In *ACNS (to appear)*, 2007. `http://eprint.iacr.org/2006/473`.

[18] Jens Groth. Rerandomizable and replayable adaptive chosen ciphertext attack secure cryptosystems. In *TCC '04*, pages 152–170, 2004.

[19] Satoshi Hada. Zero-knowledge and code obfuscation. In *ASIACRYPT '00*, volume 1976 of LNCS, pages 443–457, 2000.

[20] Susan Hohenberger, Guy N. Rothblum, abhi shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. In *TCC '07*, volume 4392 of LNCS, pages 233–252, 2007.

[21] Markus Jakobsson. On quorum controlled asymmetric proxy re-encryption. In *PKC '99*, pages 112–121, 1999.

[22] Masahiro Mambo and Eiji Okamoto. Proxy Cryptosystems: Delegation of the Power to Decrypt Ciphertexts. *IEICE Trans. Fund. Elect. Communications and CS*, E80-A/1:54–63, 1997.

[23] Manoj Prabhakaran and Mike Rosulek. Anonymous rerandomizable rcca encryption. In *CRYPTO '07 (to appear)*, 2007.

[24] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT '05*, volume 3494 of LNCS, pages 457–473, 2005.

[25] Tony Smith. DVD Jon: buy DRM-less Tracks from Apple iTunes, March 18, 2005. Available at `http://www.theregister.co.uk/2005/03/18/itunes_pymusique`.

[26] Lidong Zhou, Michael A. Marsh, Fred B. Schneider, and Anna Redz. Distributed blinding for Elgamal re-encryption. Technical Report 2004–1924, Cornell Computer Science Dept., 2004.

A UC Definitions of Security for Proxy Re-Encryption

We wish to formulate the security guarantees provided by a CCA-secure PRE scheme within the universally composable security framework. This turns out to be non-trivial; see more discussion below. We thus provide *two* such definitions, namely two formulations of an "ideal proxy re-encryption functionality", and show that one formulation implies Definition 2.5, whereas the second formulation is implied by Definition 2.5. This allows us to "sandwich" the security guarantees provided by Definition 2.5, when formulated in a composable, simulation based framework.

We first describe the strong formulation, denoted $\mathcal{F}_{\mathtt{sPRE}}$. We extend the ideal public-key encryption functionality $\mathcal{F}_{\mathtt{PKE}}$ from [9] to handle re-encryption. To do so, we must allow a single instance of $\mathcal{F}_{\mathtt{sPRE}}$ to handle multiple pairs of encryption and decryption keys, as well as multiple re-encryption proxies. This is in contrast to the case of $\mathcal{F}_{\mathtt{PKE}}$, where each instance handles only a single pair of encryption and decryption keys. In addition, in order to simplify the presentation, we assume that the size of the domain D of plaintexts is super-polynomial. (When D is small, the present formulation becomes unnecessarily restrictive. This restrictiveness can be disposed of at the price of a more cumbersome formulation.)

More specifically, as in the case of $\mathcal{F}_{\mathtt{PKE}}$, $\mathcal{F}_{\mathtt{sPRE}}$ provides a key generation interface for primary encryption and decryption keys, where the keys themselves are formal ("dummy") values chosen by the adversary, the encryption key (algorithm) is returned to the registering party, and the decryption key (algorithm) is locally recorded. In addition, there is a re-encryption key generation interface, where two registered parties P and P' can provide a proxy X with the formal capability to re-encrypt ciphertexts from P to P' and vice versa. As part of this operation the adversary provides $\mathcal{F}_{\mathtt{sPRE}}$ with a "dummy" re-encryption key $e_{P,P'}$ between P and P'. These interfaces also allow the adversary to determine the identities of corrupted parties and proxies, with effects described below. It is assumed that the adversary specifies parties as corrupted only before the encryption interface is used for the first time. This manifests the restriction of the modeling to non-adaptive party corruption.

We say that a party P is effectively corrupted if it is corrupted, or if it has set a re-encryption key with a party P' that is either corrupted or effectively corrupted, and in addition the re-encryption proxy for P, P' is corrupted. Essentially, if a party P is effectively corrupted then the adversary can "by definition of the problem" freely learn any message encrypted to P. Indeed, $\mathcal{F}_{\mathtt{sPRE}}$ will not guarantee secrecy for effectively corrupted parties.

The encryption interface allows arbitrary encryptors to "formally encrypt" messages, namely to obtain "formal ciphertexts" that will decrypt to the correct message, while guaranteeing unconditional secrecy *as long as the target party is not effectively corrupted.* This is done as follows: If the target party P is not effectively corrupted, then the formal ciphertext c for a message m is computed by applying the dummy encryption algorithm provided by the adversary to a random message r in the domain. Then, the tuple (P, m, r, c) is recorded. Recording the random value r is done in order to correctly decrypt re-encrypted ciphertexts (see details below). If the target party is effectively corrupted then c is computed by applying the dummy encryption algorithm to the actual message m to be encrypted. This represents the fact that no secrecy is guaranteed in this case.

In addition to encryption, $\mathcal{F}_{\texttt{sPRE}}$ provides a re-encryption interface. This interface is simple, and allows a re-encryption proxy between P and P' to transform a dummy ciphertext c to a dummy ciphertext c' in a way that guarantees that when P' decrypts c' it will obtain the same value as that obtained by P when decrypting c. The value of c' is computed by applying the recorded algorithm $e_{P,P'}$ to c.

The decryption interface allows a registered party P to decrypt ciphertexts c addressed to it. The decryption values are computed as follows. First, $\mathcal{F}_{\texttt{sPRE}}$ checks whether there is a recorded tuple which contains ciphertext c and target party P. If so, the corresponding plaintext m is returned. This case guarantees correct decryption for messages that were legitimately encrypted and re-encrypted using $\mathcal{F}_{\texttt{sPRE}}$'s interfaces. Next, $\mathcal{F}_{\texttt{sPRE}}$ "helps" the adversary (namely, the simulator) by proceeding as follows. First, the value $r = d_p(c)$ is computed, where d_P is the dummy decryption algorithm provided by the adversary when P registered. Next, if there is any recorded tuple where the third field (i.e., the random value chosen at encryption time) is r, then the corresponding plaintext m from that tuple is returned. This represents the case where c was generated by applying the re-encryption algorithm not via the interface provided by $\mathcal{F}_{\texttt{sPRE}}$. (This situation is possible when a re-encryption proxy gets corrupted and its key becomes exposed. In this case we'd like to allow the adversary to create new ciphertexts that decrypt to the same value as existing ones, while still maintaining the secrecy of the decrypted value. This situation is somewhat reminiscent of the case of RCCA security. We note that for this step to be effective, the domain D of plaintexts should be super-polynomial in order to avoid collisions.) Finally, if no matching tuple is found, $\mathcal{F}_{\texttt{sPRE}}$ outputs r as the decryption value. This step is similar to the case of plain $\mathcal{F}_{\texttt{PKE}}$, and provides assistance to the simulator in the case where the ciphertext c was simply generated by the environment as an encryption of r to party P, not via the legitimate encryption interface.

As stated more precisely below, functionality $\mathcal{F}_{\texttt{sPRE}}$ captures the same security notion as Definition 2.5, except for the following issue: Consider a party P that's not effectively corrupted, but has set a re-encryption proxy X with a corrupted party P'. (That is, X is not corrupted and the re-encryption key between P and P' is secret.) Now, assume that a ciphertext c, that was generated for P as an encryption of a message m, is being re-encrypted to a ciphertext c' for P'. Recall that, since P is not effectively corrupted, the ciphertext c is a "dummy ciphertext" that's statistically unrelated to m. In contrast, c' must be a value that decrypts to m under a key that's known to the adversary and environment. Furthermore, c' must look like a "plausible re-encryption" of c. Thus, a scheme that realizes $\mathcal{F}_{\texttt{sPRE}}$ must allow a "simulator" to "convincingly" transform a ciphertext c that's unrelated to m into a ciphertext c' linked to m. This is a property that seems to be an artifact of the simulation paradigm; in particular, it is not implied by Definition 2.5. In particular, our scheme does not have this property.

The weaker proxy-re-encryption ideal functionality, $\mathcal{F}_{\texttt{wPRE}}$, bypasses this problem by considering any party that has a re-encryption key with an effectively corrupted party to be effectively corrupted, regardless of whether the relevant proxy is corrupted. This provides a somewhat weaker security guarantee, but the gain is that we can now show that this notion is implied by Definition 2.5, thus providing a "lower bound" of the security provided by Definition 2.5 in a simulation-based framework. Figure 1 presents the re-encryption functionalities, $\mathcal{F}_{\texttt{sPRE}}$ and $\mathcal{F}_{\texttt{wPRE}}$.

Relations with Definition 2.5. To formalize the relationship between realizing $\mathcal{F}_{\texttt{sPRE}}, \mathcal{F}_{\texttt{wPRE}}$ and Definition 2.5, we first describe a natural transformation from a PRE scheme to a protocol geared towards realizing $\mathcal{F}_{\texttt{sPRE}}, \mathcal{F}_{\texttt{wPRE}}$. We only consider security of the protocol in a non-adaptive setting, where identities of the corrupted parties, including the corrupted proxies, are fixed in advance. Formally, this is captured by defining the protocol so that corruption requests by the adversary are

Functionality $\mathcal{F}_{\mathtt{sPRE}}$ (with message domain D)

Key Generation: When receiving (KeyGen, sid) from some party P, send $(\mathsf{KeyGen}, sid, P)$ to the adversary. When receiving algorithms (e_P, d_P) and a "corrupted" bit from the adversary, register (P, e_P, d_P) and output $(\mathsf{EncryptionAlgorithm}, sid, e_P)$ to P. In addition, if the "corrupted" bit is set then record P as corrupted.

Re-Encryption Key Generation: When receiving $(\mathsf{ReKeyGen}, sid, P, P', X)$ from registered parties P and P', send $(\mathsf{ReKeyGen}, sid, P, P', X)$ to the adversary. When receiving algorithm $e_{P,P'}$ and a "corrupted" bit from the adversary, record $(P, P', X, e_{P,P'})$ and $(P', P, e_{P,P'})$. Output $(\mathsf{Proxy}, sid, P, P')$ to party X (the proxy). In addition, if the "corrupted" bit is set then record X as corrupted.

A registered party P is called effectively corrupted if it is corrupted, or it has registered a re-encryption key with an effectively corrupted party where the proxy is corrupted.

Encryption: When receiving $(\mathsf{Enc}, sid, m, e)$ from some party E, do: If m is not in the legitimate encryption domain, output an error message. Else, if $e = e_P$ for some registered party P which is not effectively corrupted, then choose a random message $r_m \leftarrow D$, run algorithm $e_P(r_m)$ (making the necessary random choices) and let c be the outcome. Else (i.e., P is either effectively corrupted or not registered), let $c = e(m)$, and let $r_m = \perp$ where $\perp \notin D$. In either case, record (P, m, r_m, c) and output $(\mathsf{Ciphertext}, sid, c)$ to E.

Re-Encryption: When receiving $(\mathsf{ReEnc}, sid, c, e_P, e_{P'})$ from party X, where $(P, P', X, e_{P,P'})$ is recorded, compute $c' = e_{P,P'}(c)$, and return $(\mathsf{Ciphertext}, sid, c')$ to X. In addition, if there is a record (P, m, r_m, c) then add the record (P', m, r_m, c').

Decryption: When receiving (Dec, sid, c) from a registered party P, do: If there is a recorded entry (P, m, r, c) for some m and r, then set $\mu \leftarrow m$. Else, compute $r = d_P(c)$. If there is a recorded entry (P', m', r, c') for some P', m', c', then set $\mu \leftarrow m'$. (If there is more than a single such entry then output an error message.) Else set $\mu \leftarrow r$. Return $(\mathsf{Plaintext}, sid, \mu)$ to P'.

Figure 1: The strong version of the ideal functionality for (bidirectional) proxy re-encryption, $\mathcal{F}_{\mathtt{sPRE}}$. The weak version, $\mathcal{F}_{\mathtt{wPRE}}$, is identical except that a party is called effectively corrupted whenever it has registered a re-encryption key with an effectively corrupted party, even when the proxy is not corrupted. The unidirectional case is captured by modifying the re-encryption key generation interface so that it registers proxies with the ability to re-encrypt from P to P', and not necessarily vice versa.

honored only at the onset of the computation, before any encryption request is made.

Let $\Sigma = (\mathsf{KeyGen}, \mathsf{ReKeyGen}, \mathsf{Enc}, \mathsf{ReEnc}, \mathsf{Dec})$ be a bidirectional re-encryption scheme. The protocol π_Σ is defined as follows. It uses an ideal subroutine $\mathcal{F}_{\mathtt{RKG}}$ that takes as input the secret keys sk, sk' of two parties and an identity X of a proxy, and outputs to X a re-encryption key $\mathsf{ReKeyGen}(sk, sk')$.

1. On input (KeyGen, sid), party P computes $(pk, sk) \leftarrow \mathsf{KeyGen}$ and then outputs the tuple $(\mathsf{EncryptionAlgorithm}, sid, pk)$. At this point, if P gets a corruption request from the adversary then it forwards sk to the adversary. Corruption requests at a later point are ignored.
2. On input $(\mathsf{ReKeyGen}, sid, P, P', X)$, and if not corrupted, party P send $(\mathsf{ReKeyGen}, sid, P, P', X, sk)$ to $\mathcal{F}_{\mathtt{RKG}}$. When X receives output $e_{P,P'}$ from $\mathcal{F}_{\mathtt{RKG}}$, it records this value. If at this

point X gets a corruption request from the adversary then it forwards $e_{P,P'}$ to the adversary. Corruption requests at a later point are ignored.

3. On input $(\mathsf{Enc}, sid, m, e)$, party E compute $c = \mathsf{Enc}(e, m)$ and output $(\mathsf{Ciphertext}, sid, c)$.
4. On input $(\mathsf{ReEnc}, sid, c, P, P')$, proxy X computes $c' = \mathsf{ReEnc}(e_{P,P'}, c)$, and outputs $(\mathsf{Ciphertext}, sid, c')$.
5. When receiving (Dec, sid, c), party P computes $m = \mathsf{Dec}(sk, c)$, where sk is the locally recorded decryption key, and outputs m. (If there is no locally recorded key then output an error message.)

B Proof of Theorem 2.8

We show that if a PRE scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{ReKeyGen}, \mathsf{Enc}, \mathsf{ReEnc}, \mathsf{Dec})$ is strongly UC-secure (i.e., the protocol π_Σ UC-realizes $\mathcal{F}_{\mathtt{sPRE}}$ for to domain D and non-adaptive corruptions) then it is $\mathtt{PRE}$-CCA secure for domain D. This part of the proof holds for domains D of any size. The proof follows the same general outline as the proof of equivalence between the game-based and UC formulations for RCCA security [12].

Assume that there exists an adversary F that wins the $\mathtt{PRE}$-CCA game against Σ with advantage ε. We construct an environment $\mathcal{E}$ that distinguishes with probability at least ε between an interaction with $\mathcal{F}_{\mathtt{sPRE}}$ with *any* adversary $\mathcal{S}$, and a real-life interaction with the dummy adversary and $\pi = \pi_\Sigma$. Environment $\mathcal{E}$ invokes a copy of $\mathbb{F}$, and proceeds as follows:

1. Invocations by F of the corrupted (resp., uncorrupted) key generation oracles are answered by calling new corrupted (resp., uncorrupted) parties with a key generation request, and forwarding the obtained keys to F.

2. Re-encryption key generation queries are answered by invoking the two relevant parties with requests to provide a *corrupted* proxy X with a re-encryption key between them, and forwarding the obtained key to F. (Recall that F expects to obtain re-encryption keys only between parties which are not effectively corrupted.)

3. Re-encryption queries are answered by invoking the corresponding proxy with a similar re-encryption request and forwarding the obtained ciphertext to F. (Recall that re-encryption requests can be made either between uncorrupted parties, or alternatively from an uncorrupted party to a corrupted party. Furthermore, $\mathcal{F}_{\mathtt{sPRE}}$ provides a meaningful security guarantee even in the latter case: Consider a ciphertext addressed at an uncorrupted party P. Then secrecy of the plaintext is guaranteed until the point where this ciphertext is re-encrypted to a corrupted party.)

4. The challenge query (pk^*, m_0, m_1) is answered by choosing a random bit b, invoking some party to encrypt message m_b with public key pk^*, and returning the response to F as the challenge ciphertext C^*.

5. Decryption queries (pk, C) are answered as $\mathcal{O}_{dec}$ does. That is, $\mathcal{E}$ checks, by making the necessary decryption requests to parties, if C is a derivative of C^*. If so, then $\mathcal{E}$ responds with $\perp$. Else, $\mathcal{E}$ asks the holder of key pk to decrypt C and forwards the response to F.

6. When F invokes the decision oracle with a guess bit b', if $b' = b$ then $\mathcal{E}$ outputs "real"; else it outputs "ideal".

Analyzing $\mathcal{E}$, we first claim that in an interaction of $\mathcal{E}$ with π, the view of the simulated F is identical to its view of a $\mathtt{PRE}$-CCA interaction with scheme Σ; thus, in this case F will predict the bit b with non-negligible advantage ε. Next, we claim that, in an interaction of $\mathcal{E}$ with $\mathcal{F}_{\mathtt{wPRE}}$ and *any* adversary, the view of F is *statistically independent* of the bit b; thus F cannot predict b with positive advantage.

C Proof of Theorem 2.9

We show that if a PRE scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{ReKeyGen}, \mathsf{Enc}, \mathsf{ReEnc}, \mathsf{Dec})$ is $\mathtt{PRE}$-CCA secure then it is weakly UC-secure.

Let $\pi = \pi_\Sigma$ be the protocol constructed from Σ as defined above. We show that π securely realizes $\mathcal{F}_{\mathtt{wPRE}}$. That is, we construct an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{E}$ can tell with non-negligible probability whether it interacts with $\mathcal{S}$ and the ideal protocol for $\mathcal{F}_{\mathtt{wPRE}}$, or with π and an adversary $\mathcal{A}$. (In fact, as shown in [9], it suffices to consider the *dummy adversary* $\tilde{\mathcal{A}}$ which fully follows the instructions of the environment.)

Recall that the only possible activities of $\mathcal{S}$ are, first, to give $\mathcal{F}_{\mathtt{wPRE}}$ encryption and decryption algorithms (e_P, d_P) for each registering party P, and second, to give $\mathcal{F}_{\mathtt{wPRE}}$ a re-encryption key $e_{P,P'}$ for each pair of parties P and P' that request it. These activities are carried out as follows:

1. To provide algorithms (e_P, d_P) for a registering party P, first run $(pk, sk) \leftarrow \mathsf{KeyGen}$. Algorithm $d_P(\cdot)$ is the decryption algorithm $\mathsf{Dec}(sk, \cdot)$. Algorithm $e_P(m)$ proceeds as follows: If $m =\perp$ then choose a random message $\alpha \leftarrow D$ and run $\mathsf{Enc}(pk, \alpha)$. else, return $\mathsf{Enc}(pk, \alpha)$.
2. To provide a re-encryption algorithm for parties P, P', recall the corresponding secret keys $sk_P, sk_{P'}$, compute $rk_{P\leftrightarrow P'} = \mathsf{ReKeyGen}(sk_P, sk_{P'})$ and return algorithm $\mathsf{ReEnc}(rk_{P\leftrightarrow P'}, \cdot)$.

Analyzing $\mathcal{S}$, assume for contradiction that there is an environment $\mathcal{E}$ that distinguishes between the real and ideal interactions. We use $\mathcal{E}$ to construct an adversary F that breaks the $\mathtt{PRE}$-CCA security of scheme Σ. More precisely, assume that for some value of the security parameter k we have $\text{EXEC}_{\mathcal{F}_{\mathtt{wPRE}},\mathcal{S},\mathcal{E}}(k) - \text{EXEC}_{\pi,\tilde{\mathcal{A}},\mathcal{E}}(k) > \varepsilon$. We show that F guesses the bit b correctly in the $\mathtt{PRE}$-CCA game with probability at least $1/2 + e/4p$, where p is the total number of messages encrypted throughout the run of the system. (Without loss of generality, we assume that in each execution of the protocol $\mathcal{E}$ asks to encrypt exactly p messages.)

Adversary F proceeds as follows. F first randomly chooses a number $h \leftarrow \{1, ..., p\}$. Next, F runs $\mathcal{E}$ on the following simulated interaction with a system running π (and the dummy adversary $\tilde{\mathcal{A}}$). Let m_i denote the ith message that $\mathcal{E}$ asks to encrypt in an execution.[7]

1. When $\mathcal{E}$ activates some *uncorrupted* party P with input (KeyGen, sid), then F calls the uncorrupted key generation oracle, obtains a key pk and forwards pk to $\mathcal{E}$.

2. When $\mathcal{E}$ activates some *corrupted* party P with input (KeyGen, sid), then F calls the corrupted key generation oracle, obtains a key pair (sk, pk) and forwards (sk, pk) to $\mathcal{E}$.

3. When $\mathcal{E}$ activates *uncorrupted* parties P and P' with input $(\mathsf{ReKeyGen}, sid, P, P', X)$, invoke the re-encryption key generation oracle $\mathcal{O}_{rkey}$ on input $pk_P, pk_{P'}$ and record the returned value $rk_{P\leftrightarrow P'}$ to $\mathcal{E}$. If X is corrupted then also return the value $rk_{P\leftrightarrow P'}$ to $\mathcal{E}$.

4. Encryption queries are handled as follows:

   (a) For the first $h-1$ times that $\mathcal{E}$ asks to encrypt some message, m_i, with key e_i, F returns to $\mathcal{E}$ the ciphertext $C_i = \mathsf{Enc}(e_i, m_i)$.

[7]Without loss of generality we assume that $\mathcal{E}$ only asks to encrypt messages with registered public keys e_P where P is registered and uncorrupted. Indeed, when $\mathcal{E}$ asks to encrypt a message m with another public key e, it receives a value $c = enc_e(m)$ that it can compute (or, sample) by itself.

(b) At the hth time that $\mathcal{E}$ asks to encrypt a message, m_h, with key e_h, F queries its challenge oracle with the pair of messages (e_h, m_h, r_h) where r_h is chosen at random from the domain D, obtains the test ciphertext C_h, and forwards C_h to $\mathcal{E}$. Record the tuple (e_h, m_h, r_h, C_h).

(c) For the remaining $p - h$ times that $\mathcal{E}$ asks to encrypt some message, m_i, with key e_i, F lets the encrypting party return $C_i = \mathsf{Enc}(e_i, r_i)$ where r_i is chosen at random from D. Record the tuple (e_i, m_i, r_i, C_i).

5. When $\mathcal{E}$ activates party X with input $(\mathsf{ReEnc}, sid, c, e_P,\ e_{P'})$, where neither of P, P', X are corrupted, compute $C' = \mathsf{ReEnc}(rk_{P \leftrightarrow P'}, C)$ and return C' to $\mathcal{E}$. Record the tuple $(e_{P'}, m_i, r_i, C')$.

6. When $\mathcal{E}$ activates party P with input (Dec, sid, C), do:

   (a) If P is corrupted, return $m = \mathsf{Dec}(sk_P, C)$ to $\mathcal{E}$, where sk_P is the decryption key generated for P.

   (b) Else, invoke the decryption oracle $\mathcal{O}_{dec}$ with input (pk_P, C), and let r^* denote the response. (If the response is $\perp$ then let $r^* = r_h$). Next, if there is a recorded tuple (P, m, r^*, C), return m to $\mathcal{E}$. else, return r^* to $\mathcal{E}$. (The rationale here is as follows: As long as the ideal encryption procedure does not choose the same plaintext r twice, the above procedure perfectly mimics the behavior of $\mathcal{F}_{\texttt{wPRE}}$.)

7. When $\mathcal{E}$ halts, F outputs whatever $\mathcal{E}$ outputs and halts.

Analyzing the success probability of F is done via a hybrid argument. Let the random variable H_i denote the output of $\mathcal{E}$ from an interaction that is identical to an interaction with $\mathcal{S}$ $\mathcal{F}_{\texttt{wPRE}}$, with the exception that the first i ciphertexts are computed as an encryption of the real plaintexts, rather than encryptions of random values.

Let B denote the event that two activations of the encryption interface of $fprew$ choose the same random value r. It can be seen that, conditioned on the event that B never occurs, H_0 is distributed identically to the output of $\mathcal{E}$ in the ideal interaction. Also, H_p is statistically close to the output of $\mathcal{E}$ in the interaction with Σ (this follows from the correctness of Σ). Furthermore, in a run of F, if the value C_h that F obtains from its challenge oracle is an encryption of m_h then the output of the simulated $\mathcal{E}$ has the distribution of H_{h-1}. If C_h is an encryption of r_h then the output of the simulated $\mathcal{E}$ has the distribution of H_h. Since $\mathcal{E}$ distinguishes between H_h and H_{h-1} with probability $\varepsilon/2p$, we have that F predicts the hidden bit b with advantage $\varepsilon/4p$.

Note that we made crucial use of the distinction between $\mathcal{F}_{\texttt{wPRE}}$ and $\mathcal{F}_{\texttt{sPRE}}$, more specifically of the fact that the environment can never ask to transform a ciphertext directed at an uncorrupted party into a ciphertext directed to a corrupted party.