Key-Private Proxy Re-Encryption

Giuseppe Ateniese* Karyn Benson* Susan Hohenberger*

January 22, 2009

Abstract

Proxy re-encryption (PRE) allows a proxy to convert a ciphertext encrypted under one key into an encryption of the same message under another key. The main idea is to place as little trust and reveal as little information to the proxy as necessary to allow it to perform its translations. At the very least, the proxy should not be able to learn the keys of the participants or the content of the messages it re-encrypts. However, in all prior PRE schemes, it is easy for the proxy to determine between which participants a re-encryption key can transform ciphertexts. This can be a problem in practice. For example, in a secure distributed file system, content owners may want to use the proxy to help re-encrypt sensitive information *without* revealing to the proxy the *identity* of the recipients.

In this work, we propose key-private (or anonymous) re-encryption keys as an additional useful property of PRE schemes. We formulate a definition of what it means for a PRE scheme to be secure and key-private. Surprisingly, we show that this property is not captured by prior definitions or achieved by prior schemes, including even the secure obfuscation of PRE by Hohenberger et al. (TCC 2007). Finally, we propose the first key-private PRE construction and prove its CPA-security under a simple extension of Decisional Bilinear Diffie Hellman assumption and its key-privacy under the Decision Linear assumption in the standard model.

1 Introduction

In many applications, data protected under one public key pk_1 needs to be distributed to a user with a different public key pk_2. It is not always practical for the owner of sk_1 to be online to decrypt these ciphertexts and then encrypt these contents anew under pk_2. For example, Alice might wish to have her mail server forward her encrypted email to Bob while she is on vacation. However, how can Alice do this without revealing her sk_1 to either her mail server or Bob?

As a solution to this key management problem, the concept of proxy re-encryption (PRE) was introduced [4]. Proxy re-encryption is a cryptosystem with the special property that a proxy, given special information, can efficiently convert a ciphertext for Alice into a ciphertext of the same message for Bob. The proxy should not, however, learn either party's secret key or the contents of the messages it re-encrypts. The main idea is to place as little trust in the proxy as possible. When PRE is used for distributed file systems [1], this absence of trust directly reduces the desirability for an adversary to compromise the distribution server, without compromising functionality.

In addition to hiding the contents of files from the proxy, it is also useful in practice to suppress as much meta-data as possible. For example, we might want the proxy file server to re-encrypt

*Johns Hopkins University, ateniese@cs.jhu.edu, kbenson5@jhu.edu, susan@cs.jhu.edu.

sensitive files for certain recipients *without* revealing to the proxy the recipient's identity. For example, the server might be told to re-encrypt all category one files with key one and category two files with keys two and three, without the proxy being able to deduce the public keys behind these values. This way, if the proxy is compromised, the adversary will not be able to extract a list of "who was speaking privately with whom". This is highly desirable for many encrypted communication scenarios.

This level of privacy for standard encryption schemes was formalized as *key-private* (or anonymous) encryption in 2001 by Bellare, Boldyreva, Desai and Pointcheval (BBDP) [3]. Intuitively, they studied encryption schemes where it is impossible to derive the recipient of a message from the ciphertext and the set of public keys. Consequently, the ciphertext is anonymous; that is, it cannot be linked to a particular public key and its owner. Fortunately, most public key encryption schemes already satisfy this property, such as Elgamal, Cramer-Shoup, and RSA-OAEP.

In this work, we introduce the strictly stronger notion of *key-private* (or anonymous) PRE. Intuitively, it should be impossible for the proxy and a set of colluding users to derive either the sender or receiver's identities from a re-encryption *key* even when given the public keys and flexible interaction ability within the system. As we formalize in Section 2.1, achieving key-private PRE is only possible when the underlying encryption scheme is key-private.

Unfortunately, this condition is far from sufficient. Finding a key-private PRE was a surprisingly difficult task. Whereas most standard encryption schemes are already key-private under the BBDP definition, *none* of the half-dozen existing PRE schemes are key-private under our natural definition in Section 2. This includes even the recent PRE construction of Hohenberger et al. [9], which was proven secure under a very strong *obfuscation* definition. In the next section, we discuss the problems with each existing scheme and the necessary conditions for realizing key-private PRE.

The main contribution of this work, in addition to our formal definition in Section 2, is the first realization of a key-private PRE scheme. Our construction is efficient, reasonably simple, and secure under basic assumptions about bilinear groups in the standard model. Formally, it is a unidirectional, single-hop, CPA-secure PRE with key-privacy. Thus, we show, for the first time, that this natural extension of anonymous encryption is practical and available for many existing PRE applications, as discussed in Section 1.2.

1.1 The Notion of Key-Private PRE and Prior Constructions

In this section, we examine the half-dozen existing proxy re-encryption schemes and discuss why they do not satisfy the notion of key-privacy. Let us first sketch the privacy notion wanted. Intuitively, we want to capture the strong guarantee that even an active proxy colluding with a set of malicious users in the system cannot learn from the re-encryption key the identity of the involved participants nor the contents of their encrypted messages.

Informally, the key-privacy game works as follows. First, the adversary is given the public keys of all honest users and the keypairs of all corrupt users. Next, the adversary is allowed to query two oracles an arbitrary number of times. The adversary may either request: (1) to have a chosen ciphertext under any user i re-encrypted to any user j or (2) to obtain the re-encryption key that translates ciphertexts from any user i to any user j. These oracles will operate regardless of the corruption status of i or j. Finally, the adversary must output a challenge pair of honest users (i^*, j^*), with the restriction that the adversary cannot have asked for this key before. The challenger will then return *either* the re-encryption key from i^* to j^* or a random key in the key space. The adversary wins if he can distinguish these cases with non-negligible probability.

Before discussing the problems with specific PRE constructions, let's get a better sense of what *cannot* possibly work. In Section 2.1, we point out that no *deterministic* re-encryption algorithm can satisfy the key-privacy definition. To see this, consider the generic attack where an adversary asks for a re-encryption of ciphertext C under user i to user j to obtain output C'. The adversary can then challenge on (i, j) and apply the returned re-encryption key to C. Since the re-encryption algorithm is deterministic, this should result in output C' if this is a proper key from i to j and is unlikely to do so for a random key. Unfortunately, the first four (out of six) prior PRE schemes have deterministic re-encryption algorithms, and thus cannot be key-private.

Similarly, in Section 2.1, we show that for a PRE scheme to be key-private (that is, one cannot distinguish the participants from seeing the *key*), the underlying encryption scheme must also be key-private in the sense of Bellare, Boldyreva, Desai and Pointcheval [3] (that is, one cannot distinguish the recipient from seeing the *ciphertext*). Some of the schemes also fail to have this property; mainly because they are in a bilinear setting, where the map can be used for this test.

Let us now discuss some specifics of prior schemes.

BBS PRE. Proxy re-encryption was first proposed by Blaze, Bleumer, and Strauss (BBS) [4] in Eurocrypt 1998. Their scheme, based on Elgamal, works in a group G of prime order p. Anyone can send a message $m \in G$ to user A with public key g^a (with $g \in G$) by computing $(mg^k, (g^a)^k)$, for a random $k \in \mathbb{Z}_p$. A can delegate to B (with public key g^b) her decryption rights by giving the proxy the value b/a mod p. All ciphertexts for A can be converted to ciphertexts for B by computing $(g^{ak})^{b/a} = g^{bk}$ and then releasing the ciphertext (mg^k, g^{bk}). Unfortunately, this scheme is trivially not key-private, because its re-encryption algorithm is deterministic. But there is an even easier attack: the adversary challenges on (A, B) to obtain challenge key r, this key is correct iff $r = b/a$. Using the public keys (g^a, g^b), the adversary can test this as $(g^a)^r = g^b$.

AFGH PRE. Ateniese, Fu, Green and Hohenberger [1] proposed new PRE schemes that employ bilinear pairings. Their protocols are unidirectional (a re-encryption key from A to B does not imply a key from B to A), an improvement over BBS where the keys are bidirectional. Their schemes require a bilinear map $\mathbf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, where $g \in \mathbb{G}$ and $Z = \mathbf{e}(g, g) \in \mathbb{G}_T$. In their first scheme, public key for A is g^a and similarly B's public key is g^b. The re-encryption key $rk_{A \rightarrow B}$ is $g^{b/a}$. However, this scheme is not key-private, because the adversary can challenge on (A, B) to obtain key r and then test if $r = g^{b/a}$ as $\mathbf{e}(g^a, r) = \mathbf{e}(g^b, g)$. A similar attack also works for their second scheme.[1] But since both schemes are deterministic, the generic attack also applies here.

CH PRE. Canetti and Hohenberger [7] proposed the first CCA-secure bidirectional PRE scheme in the standard model. However, even CCA-security does not ensure key-privacy, because the public keys (e.g., g^a, g^b) and re-encryption keys (e.g., b/a) are the same as in the BBS PRE, so the proxy can attack key-privacy here using the same algorithm from BBS. Part of the re-encryption algorithm of this scheme is also deterministic, and therefore, the generic attack again applies.

LV PRE. Libert and Vergnaud [11] proposed the first CCA-secure unidirectional PRE scheme in the standard model. To achieve CCA-security, they employ a quite interesting technique whereby

[1] To see why the second AFGH scheme is not key-private, consider the following attack. The adversary can ask for the re-encryption key from (C, A) to obtain $r_1 = g^{a_2 c_1}$. The adversary can next challenge on (C, B) to obtain r_2. Then the adversary can test if $r_2 = g^{b_2 c_1}$, making it a valid re-encryption key from C to B, via $\mathbf{e}(r_1, g^{b_2}) = \mathbf{e}(r_2, g^{a_2})$ with public key values g^{a_2} and g^{b_2}. This test determines if two keys have the same delegator, which is not possible under our key-privacy definition.

the encryption of the scheme in [1] is randomized by the proxy via a blinding factor that effectively hides the re-encryption key within the re-encryption (which is also followed by a proof of consistency). Interestingly, their scheme is not key-private even though the re-encryption algorithm is probabilistic. Indeed, A and B have respectively public keys g^a and g^b, and the proxy key is $rk_{A\to B} = g^{b/a}$, just as in AFGH. Thus, as in AFGH [1], the adversary can challenge on (A, B) to obtain key r and then test if $r = g^{b/a}$ as $\mathbf{e}(g^a, r) = \mathbf{e}(g^b, g)$.

HRSV PRE. Recently, Hohenberger, Rothblum, shelat, and Vaikuntanathan [9] presented a CPA-secure unidirectional PRE in the standard model, with probabilistic algorithms for performing encryption and generating re-encryption keys. Moreover, HRSV satisfied a very strong security notion, treating the re-encryption key together with the re-encryption algorithm as an *obfuscated* re-encryption program. That is, a program whose code is scrambled in such a way that: (1) it still produces the correct outputs, and yet (2) it is not possible to "reverse engineer" the program to learn its secrets (i.e., anything that cannot be learned from black-box access to the program.) Interestingly, even their strong obfuscation definition does not imply key privacy and their construction does not satisfy our definition. To see this, recall that their construction is set in a bilinear group, where Alice's public key is of the form (g, g^{a_1}, g^{a_2}) and Bob's public key is of the form (h, h^{b_1}, h^{b_2}) for random $g, h \in \mathbb{G}$ and random exponents a_1, a_2, b_1, b_2. Given these public keys, the adversary can ask to see the re-encryption key for (A, B) which will be $(y, y^{b_1/a_1}, y^{b_2/a_2})$, where $y \in \mathbb{G}$ is chosen randomly. The adversary can then challenge on (B, A) to obtain a key (r, r_1, r_2), which if correct, is of the form $(r, r^{a_1/b_1}, r^{a_2/b_2})$ for a random $r \in \mathbb{G}$. The adversary can then test for correctness as $\mathbf{e}(y, r) = \mathbf{e}(y^{b_1/a_1}, r_1) = \mathbf{e}(y^{b_2/a_2}, r_2)$. Thus, even this obfuscation is not key-private. Indeed, our notion seems difficult to satisfy because, unlike the obfuscation definition, we will allow the adversary broad query powers and the ability to collude with system users, whereas the current definition of obfuscation considers the release of only one re-encryption program.

1.2 Applications

PRE has been proposed for use in email-forwarding [4], secure file systems [1], DRM [13], and secure mailing lists [10]. All these applications can benefit from the key-privacy property in some way. In email-forwarding, Alice may not want the mail server to know to whom she is delegating her decryption rights. This is similar in the real world to a P.O. Box address where mail can be sent to a physical location but neither the sender nor the carrier may know who the actual recipient is. Alice can hide the fact that Bob is a delegatee by instructing the server to convert her encrypted emails via a key-private PRE scheme and to forward the results to an anonymous (or group) email address (i.e., an address reachable by Bob but that does not contain any identifiable information on Bob, like a P.O. Box address indeed).

In a distributed file system, PRE schemes can be used as an access control mechanism to specify who can access and read encrypted files [1]. Alice may want Bob to read some of her encrypted files, thus she instructs the file system to convert those files using a proxy re-encryption key from Alice to Bob. In a distributed file system, anyone can access those files but only Bob can read them. If the PRE scheme employed is key-private, nobody can even tell who can access and read any file in the system.

In [13], Taban, Cárdenas, and Gligor describe a secure and interoperable digital rights management (DRM) system based on proxy re-encryption and proxy re-signatures [2]. They specify, implement, and analyze a framework within which different DRM systems can interoperate. Proxy

re-encryption is used by a Domain Interoperability Manager (DIM) that translates DRM packaged digital content between devices with distinct DRM systems. The DIM is a semi-trusted entity that is susceptible to compromise, thus encryption is used to ensure privacy of the content and licenses associated with each DRM system. A key-private PRE scheme would also hide the associations between the various devices and their respective DRM systems in case of compromise.

In [10], Khurana, Heo, and Pant propose to use proxy re-encryption for SELS (Secure Email List Services), a system that provides private email discussion lists via encryption. A list is composed of several members that exchange messages internally or with other members outside the list. To send a private message to a list (and to its members), it is enough to encrypt the message under a public-key associated with the list. A List Server (LS) uses a PRE scheme to translate that encryption into encryptions under the public keys of each member of the list, respectively. If the LS server is ever compromised, the secret keys of the list and its members would remain protected as well as the content of any messages exchanged within the list. However, the identities of the members in a list would be exposed by just looking at the re-encryption keys. This may not be desirable in many contexts and thus a key-private PRE scheme would be preferable whenever the privacy of list members must be guaranteed.

In [12], Suriadi, Foo and Smith use proxy re-encryption to develop a credential system with conditional privacy. Their system has many proxies providing keys to parities who wish to remain *anonymous*. They use multiple-hops in their key distribution to help maintain anonymity; it would be possible to instead use a key-private PRE scheme.

2 Key-Private PRE Definitions

We build upon the re-encryption definitions of [1] and [7] to introduce the concept of key privacy. We begin by specifying the input/output behavior of a proxy re-encryption scheme. For simplicity, we will only consider a definition for unidirectional, single-hop PREs. By single-hop, we mean that only original ciphertexts (and not re-encrypted ciphertexts) can be re-encrypted. One could extend the basic notions of our security and key privacy definitions for multi-hop schemes, but this requires a more involved definition of when users become corrupt, and may be more appropriate for CCA-secure schemes.

Definition 2.1 ***(Unidirectional, Single-Hop PRE)*** *A unidirectional, single-hop, proxy re-encryption scheme is a tuple of algorithms* $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ReKeyGen}, \mathsf{Enc}, \mathsf{ReEnc}, \mathsf{Dec})$ *for message space* M*:*

- $\mathsf{Setup}(1^k) \rightarrow PP$*. On input security parameter* 1^k*, the setup algorithm outputs the public parameters* PP*.*
- $\mathsf{KeyGen}(PP) \rightarrow (pk, sk)$*. On input public parameters, the key generation algorithm* KeyGen *outputs a public key* pk *and a secret key* sk*.*
- $\mathsf{ReKeyGen}(PP, sk_i, pk_j) \rightarrow rk_{i \rightarrow j}$*. Given a secret key* sk_i *and a public key* pk_j*, where* $i \neq j$*, this algorithm outputs a unidirectional re-encryption key* $rk_{i \rightarrow j}$*. The restriction that* $i \neq j$ *is provided as re-encrypting a message to the original recipient is impractical.*
- $\mathsf{Enc}(PP, pk_i, m) \rightarrow C_i$*. On input a public key* pk_i *and a message* $m \in M$*, the encryption algorithm outputs an original ciphertext* C_i*.*
- $\mathsf{ReEnc}(PP, rk_{i \rightarrow j}, C_i) \rightarrow C_j$*. Given a re-encryption key from* i *to* j *and an original ciphertext for* i*, the re-encryption algorithm outputs a ciphertext for* j *or the error symbol* $\perp$*.*

- $\mathsf{Dec}(PP, sk_i, C_i) \to m$. *Given a secret key for user i and a ciphertext for i, the decryption algorithm* Dec *outputs a message $m \in M$ or the error symbol $\perp$.*

A unidirectional, single-hop PRE scheme Π is correct with respect to domain M if:

- For all $(pk, sk) \in \mathsf{KeyGen}(PP)$ and all $m \in M$, it holds that $\mathsf{Dec}(PP, sk, \mathsf{Enc}(PP, pk, m)) = m$.
- For all pairs $(pk_i, sk_i), (pk_j, sk_j) \in \mathsf{KeyGen}(PP)$ and keys $rk_{i \to j} \in \mathsf{ReKeyGen}(PP, sk_i, pk_j)$, and $m \in M$, it holds that $\mathsf{Dec}(PP, sk_j, \mathsf{ReEnc}(PP, rk_{i \to j}, \mathsf{Enc}(PP, pk_i, m))) = m$.

Next, let's turn to the standard security definition, simplifying the presentation of [1].

Definition 2.2 ***(Unidirectional, Single-Hop PRE CPA-Security Game)*** *Let 1^k be the security parameter. Let $\mathcal{A}$ be an oracle TM, representing the adversary. The PRE-CPA game consists of an execution of $\mathcal{A}$ with the following oracles. The game consists of three phases, which are executed in order. Within each phase each oracle can be executed in any order,* $\mathrm{poly}(k)$ *times, unless otherwise specified.*
Phase 1:

- ***Public Parameter Generation:*** *The public parameters are generated and given to $\mathcal{A}$. This oracle is executed first and only once.*
- ***Uncorrupted Key Generation:*** *Obtain a new key pair $(pk, sk) \leftarrow \mathsf{KeyGen}(PP)$. $\mathcal{A}$ is given pk. Let Γ_H be the set of honest user indices.*
- ***Corrupted Key Generation:*** *Obtain a new key pair $(pk, sk) \leftarrow \mathsf{KeyGen}(PP)$. $\mathcal{A}$ is given (pk, sk). Let Γ_C be the set of corrupt user indices.*

Phase 2:

- ***Re-encryption Key Generation $\mathcal{O}_{rkey}$:*** *On input (i, j) by the adversary, where the key pairs for i and j were generated in Phase 1, return the key $rk_{i \to j} = \mathsf{ReKeyGen}(PP, sk_i, pk_j)$. All requests where $i = j$ or where $i \in \Gamma_H$ and $j \in \Gamma_C$ are ignored (an output of $\perp$).*
- ***Re-encryption $\mathcal{O}_{renc}$:*** *On input (i, j, C_i) where the key pairs for i and j were generated in Phase 1, return $C_j = \mathsf{ReEnc}(PP, \mathsf{ReKeyGen}(PP, sk_i, pk_j), C_i)$. All requests where $i = j$ or where $i \in \Gamma_H$ and $j \in \Gamma_C$ are ignored (an output of $\perp$).*
- ***Challenge Oracle:*** *On input (i, m_0, m_1), the oracle chooses a random $b \leftarrow \{0, 1\}$ and returns the challenge ciphertext $C_i = \mathsf{Enc}(PP, pk_i, m_b)$. This oracle can only be queried once, and it is required that $i \in \Gamma_H$.*

Phase 3:

- ***Decision:*** *Eventually, $\mathcal{A}$ outputs decision $b' \in \{0, 1\}$. $\mathcal{A}$ wins the game if and only if $b = b'$.*

Definition 2.3 ***(Unidirectional, Single-Hop PRE CPA Security)*** *Given security parameter 1^k, a PRE scheme is Unidirectional PRE CPA secure for domain M of messages if is it correct for M and $\forall$ p.p.t. adversaries $\mathcal{A}$, $\exists$ a negligible function ε such that $\mathcal{A}$ wins the unidirectional PRE-CPA game with probability at most $\frac{1}{2} + \varepsilon(k)$.*

Remark 2.4 (Corruptions) As in many prior re-encryption papers [1, 7, 11], we work in a static corruption model, where the adversary must chose to either corrupt a party or not at the time the party's keypair is generated. Indeed, the problem of handling dynamic corruptions for any

encryption scheme is a classically difficult problem. This rules out allowing the adversary to query $\mathcal{O}_{rkey}$ from an honest to a corrupt user, since this action would corrupt the honest user. Moreover, we also disallow adversarial queries to $\mathcal{O}_{renc}$ from honest to corrupt users, as in [1], since such access could simulate a decryption oracle which we do not consider in CPA-secure constructions.

Next, we turn to the issue of what it means for a re-encryption key to be key-private. Informally, we want a proxy to be unable to identify either the delegator i or the delegatee j when given the re-encryption key $rk_{i\to j}$ and flexible interaction with the system (e.g., other re-encryption keys, access to re-encryption oracles, etc.) To capture this idea, we say that the proxy is allowed to choose (i, j) and then cannot distinguish the valid key $rk_{i\to j}$ from a random value in the key space.

Our definition for key privacy is more challenging to realize than CPA-security, because most of the restrictions on how the adversary can call $\mathcal{O}_{rkey}$ and $\mathcal{O}_{renc}$ are now removed. Indeed, we allow a form of dynamic corruption here. For example, it is now legal for the adversary to ask for re-encryption keys and re-encryptions from honest to corrupt parties, and then later to challenge on these honest parties. In other words, key-privacy is maintained even when honest parties unwisely delegate decryption capabilities to corrupt parties.

Definition 2.5 ***(Unidirectional, Single-Hop PRE Key-Privacy Game)*** *Let k be the security parameter. Let $\mathcal{A}$ be an oracle TM, representing the adversary. The PRE Key-Privacy Game consists of an execution of $\mathcal{A}$ with the same oracles as before unless specified below. There are three phases.*
Phase 1:

- *The adversary is given the public parameters, and then may request uncorrupted or corrupted key pairs to be created, as before.*

Phase 2:

- ***Re-encryption Key Generation*** $\mathcal{O}_{rkey}$***:*** *On input (i, j) by the adversary, where the key pairs for i and j were generated in Phase 1, return the key in table T corresponding to (i, j). If there is no such entry in the table, compute it as* $\mathsf{ReKeyGen}(PP, sk_i, pk_j)$*, add the key to the table T in cell (i, j), and output this key. The oracle will only produce a* single *re-encryption key for (i, j). If $i = j$ then the error symbol $\perp$ is returned. Note that there is no longer the restriction that $i \notin \Gamma_H$ or $j \notin \Gamma_C$.*
- ***Re-encryption*** $\mathcal{O}_{renc}$***:*** *On input (i, j, C_i) where the key pairs for i and j were generated by* KeyGen*, obtain the re-encryption key s corresponding to (i, j) in table T. If no such key exists, create it as $s \leftarrow \mathsf{ReKeyGen}(PP, sk_i, pk_j)$ and save it in table T. Return either $C_j = \mathsf{ReEnc}(PP, s, C_i)$ or $\perp$ if $i = j$.*
- ***Challenge Oracle:*** *This oracle can only be challenged once. On input (i, j), the oracle sets s to be the key corresponding to (i, j) in table T. If no such key exists, it creates it as $s \leftarrow \mathsf{ReKeyGen}(PP, sk_i, pk_j)$. The oracle then chooses a bit $b \leftarrow \{0, 1\}$ and then returns the value s if $b = 1$ and a random key in the key space otherwise. The constraints are that $\mathcal{O}_{rkey}$ must not have been queried for (i, j) before, $i \neq j$ and $i, j \in \Gamma_H$.*

Phase 3:

- ***Decision:*** *Eventually, $\mathcal{A}$ outputs decision $b' \in \{0, 1\}$. We say that $\mathcal{A}$ wins the game if and only if $b = b'$.*

Definition 2.6 ***(Unidirectional, Single-Hop PRE Key Privacy)*** *For security parameter 1^k, a PRE scheme is key-private if $\forall$ p.p.t. adversaries $\mathcal{A}$, $\exists$ a negligible function ε such that $\mathcal{A}$ wins the unidirectional PRE Key-Privacy Game with probability at most $\frac{1}{2} + \varepsilon(k)$.*

2.1 Impossibility Results for Key-Private Re-Encryption

Before seeing the construction, we lay out some necessary, but not sufficient, conditions for satisfying the above definition in two simple lemmas.

Lemma 2.7 *Any bidirectional or unidirectional re-encryption scheme* (Setup, KeyGen, ReKeyGen, Enc, ReEnc, Dec)*, where the* ReEnc *algorithm is deterministic cannot satisfy key-privacy (Definition 2.6).*

Proof. Suppose ReEnc is deterministic. An adversary $\mathcal{A}$ wins the key-privacy game as follows:

1. Ask for a set of uncorrupted parties to be generated; obtain the public parameters and keys.
2. Choose a random m in the message space and compute $c = \mathsf{Enc}(PP, pk_1, m)$.
3. Query the re-encryption oracle $\mathcal{O}_{renc}(1, 2, c)$ to obtain the response c'.
4. Challenge on identities $(1, 2)$ and obtain the challenge key s. (Recall that, by definition, the key-privacy challenger will return the same key here that it used in the previous step.)
5. Using s, run the deterministic algorithm $\mathsf{ReEnc}(PP, s, c) \rightarrow c''$.
6. If $c' = c''$, output 1, else output 0.

It is easy to see that $\mathcal{A}$ succeeds with overwhelming probability. □

This lemma immediately rules out almost all prior PRE constructions [4, 1, 7] as candidates for key privacy. Nor is it obvious how to transform these constructions into key-private schemes. The schemes by Libert and Vergnaud [11] and Hohenberger et al. [9] employ probabilistic re-encryption algorithms, but they still admit key-privacy attacks. Thus, a probabilistic re-encryption algorithm is a necessary, but not sufficient condition.

Lemma 2.8 *Any bidirectional or unidirectional re-encryption scheme* (Setup, KeyGen, ReKeyGen, Enc, ReEnc, Dec) *satisfying the key-privacy (Definition 2.6) implies that* (Setup, KeyGen, Enc, Dec) *admits a key-private* encryption *scheme according to the standard definition [3].*

In other words, it is not possible for a PRE scheme to be key-private, unless the underlying encryption resulting from a re-encryption is key-private. The point here is that any key-private PRE must admit some key-private encryption scheme. Bellare, Boldyreva, Desai and Pointcheval [3] introduced key-private encryption, where an adversary cannot distinguish the intended recipient from the ciphertext. More formally, the adversary is given two public keys pk_0, pk_1, chooses a message m, is given an encryption of m under one of the two keys $b \in \{0, 1\}$ chosen at random, and finally issues a guess $b' \in \{0, 1\}$. The security notion requires that all efficient adversaries cannot achieve $b = b'$ with probability non-negligibly better than random guessing.

For a PRE scheme to be key-private, the proxy cannot distinguish the intended recipient from the ciphertext *even when given access to re-encryption keys and re-encryption oracles.* To see this, consider that otherwise an adversary $\mathcal{A}$ can win the key-privacy game as follows:

1. Ask for n uncorrupted parties to be generated; obtain the public parameters and keys.
2. Choose a random m in the message space and compute $c = \mathsf{Enc}(PP, pk_1, m)$.

3. Challenge on identities $(1, 2)$ and obtain the challenge key s.
4. Using s, run the possibly probabilistic algorithm $\mathsf{ReEnc}(PP, s, c) \to c'$.
5. If c' is a ciphertext under public key pk_2, output 1, else output 0.

The BBS PRE uses Elgamal (in a non-bilinear setting) as its encryption base and thus satisfies anonymous encryption via Bellare et al. [3], although it is ultimately not a key-private PRE. Thus, this condition is also necessary, but not sufficient.

3 A Key-Private PRE Scheme

3.1 Algebraic Setting

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 $(q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, where $\mathbb{G}, \mathbb{G}_T$ are of prime order $q \in \Theta(2^k)$ and $\langle g \rangle = \mathbb{G}$. The efficient mapping $\mathbf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is both: (*Bilinear*) for all $g \in \mathbb{G}$ and $a, b \in \mathbb{Z}_q$, $\mathbf{e}(g^a, g^b) = \mathbf{e}(g, g)^{ab}$; and (*Non-degenerate*) if g generates $\mathbb{G}$, then $\mathbf{e}(g, g) \neq 1$. We consider the following assumptions.

Decisional Bilinear Diffie-Hellman (DBDH) [6]: 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 $\mathcal{A}$, there exists a negligible function ε such that the following probability is less than or equal to $1/2 + \varepsilon(k)$:

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

Extended Decisional Bilinear Diffie-Hellman (EDBDH) [1]: 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 $\mathcal{A}$, there exists a negligible function ε such that the following probability is less than or equal to $1/2 + \varepsilon(k)$:

$$\Pr[a, b, c, d \leftarrow \mathbb{Z}_q;\ x_1 \leftarrow \mathbf{e}(g, g)^{abc};\ x_0 \leftarrow \mathbf{e}(g, g)^{d};\ z \leftarrow \{0, 1\}; \\ z' \leftarrow \mathcal{A}(g, g^a, g^b, g^c, \mathbf{e}(g, g)^{bc^2}, x_z) : z = z'].$$

Decision Linear [5]: Let $\mathsf{BSetup}(1^k) \to (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, where $\langle g \rangle = \mathbb{G}$. Let h, f be random generators in $\mathbb{G}$. For all p.p.t. adversaries $\mathcal{A}$, there exists a negligible function ε such that the following probability is less than or equal to $1/2 + \varepsilon(k)$:

$$\Pr[x, y, r \leftarrow \mathbb{Z}_q;\ q_1 \leftarrow f^{x+y};\ q_0 \leftarrow f^{r};\ z \leftarrow \{0, 1\}; z' \leftarrow \mathcal{A}(g, h, f, g^x, h^y, q_z) : z = z'].$$

3.2 The Construction

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

Setup (Setup): Run $\mathsf{BSetup}(1^k) \to (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$, where $\langle g \rangle = \mathbb{G}$. Choose a random generator $h \in \mathbb{G}$. Compute $Z = \mathbf{e}(g, h)$, and set the public parameters $PP = (g, h, Z)$. In the following, we assume that all parties have PP.

Key Generation (KeyGen): Choose random values $a_1, a_2 \in \mathbb{Z}_q$ and set the public key as $pk = (Z^{a_1}, g^{a_2})$ with secret key $sk = (a_1, a_2)$.

Re-Encryption Key Generation (ReKeyGen)**:** A user A with secret key (a_1, a_2) can delegate to a user B with public key (Z^{b_1}, g^{b_2}) as:

1. Select random values $r, w \in \mathbb{Z}_q$.
2. Compute $rk_{A\to B} = ((g^{b_2})^{a_1+r}, h^r, \mathbf{e}(g^{b_2}, h)^w, \mathbf{e}(g,h)^w) = (g^{b_2(a_1+r)}, h^r, Z^{b_2 w}, Z^w)$.

Encryption (Enc)**:** To encrypt a message $m \in \mathbb{G}_T$ under public key $pk_A = (Z^{a_1}, g^{a_2})$, do:

1. Select random value $k \in \mathbb{Z}_q$.
2. Compute the ciphertext $(g^k, h^k, m \cdot Z^{a_1 k})$.

We note that, as in prior schemes [1], it is possible to use this same public key to produce a ciphertext that *cannot* be re-encrypted, and thus only opened by the holder of sk_A by selecting a random $k \in \mathbb{Z}_q$ and outputting the Elgamal ciphertext $(\mathbf{e}(g^{a_2}, h)^k, m \cdot Z^k) = (Z^{a_2 k}, m \cdot Z^k)$. We refer to this as a *first-level* ciphertext, and re-encryptable ones as *second-level* ciphertexts.

Re-Encryption (ReEnc)**:** Given a re-encryption key $rk_{A\to B} = (R_1, R_2, R_3, R_4) = (g^{b_2(a_1+r)}, h^r, Z^{b_2 w}, Z^w)$, it is possible to convert a second-level ciphertext $C_A = (\alpha, \beta, \gamma)$ for A into a first-level ciphertext for B as follows:

1. Verify that the ciphertext is well-formed, by checking that it uses consistent randomness in its first two parts as: $\mathbf{e}(\alpha, h) = \mathbf{e}(g, \beta)$. If this does not hold, output $\perp$ and abort.
2. Otherwise, there exists some $k \in \mathbb{Z}_q$ and $m \in \mathbb{G}_T$ such that $\alpha = g^k$, $\beta = h^k$ and $\gamma = m \cdot Z^{a_1 k}$, and thus, (α, β, γ) is a valid encryption of this m under $pk_A = (Z^{a_1}, g^{a_2})$.
3. Compute $t_1 = \mathbf{e}(R_1, \beta) = \mathbf{e}(g^{b_2(a_1+r)}, h^k) = Z^{b_2 k(a_1+r)}$.
4. Compute $t_2 = \gamma \cdot \mathbf{e}(\alpha, R_2) = m \cdot Z^{a_1 k} \cdot \mathbf{e}(g^k, h^r) = m \cdot Z^{a_1 k} \cdot Z^{rk} = m \cdot Z^{k(a_1+r)}$.
5. Select a random $w' \in \mathbb{Z}_q$.
6. Re-randomize t_1 by setting $t_1' = t_1 \cdot R_3^{w'} = Z^{b_2 k(a_1+r)} \cdot (Z^{w b_2})^{w'} = Z^{b_2(k(a_1+r)+ww')}$.
7. Re-randomize t_2 by setting $t_2' = t_2 \cdot R_4^{w'} = m \cdot Z^{k(a_1+r)} \cdot (Z^w)^{w'} = m \cdot Z^{k(a_1+r)+ww'}$.
8. Publish $C_B = (t_1', t_2') = (Z^{b_2 y}, m \cdot Z^y)$, where $y = k(a_1 + r) + ww'$.

Decryption (Dec)**:** Given secret key (a_1, a_2), to decrypt a first-level ciphertext (α, β), compute $m = \beta / \alpha^{1/a_2}$; and to decrypt a second-level ciphertext (α, β, γ), output $\perp$ if $\mathbf{e}(\alpha, h) \neq \mathbf{e}(g, \beta)$, otherwise output $m = \gamma / \mathbf{e}(\alpha, h)^{a_1}$.

Fortunately, this scheme is practical and multi-purpose. Public keys can be used either for re-encryption purposes or for regular Elgamal encryptions. For completeness, we show in Appendix A that the CPA-security of the first-level ciphertexts holds under DBDH.

3.3 Security Analysis

We first argue that the above scheme is secure and then that it is key-private.

Theorem 3.1 (CPA Security) *Under the EDBDH assumption in $\mathbb{G}$, scheme Π is a unidirectional, single-hop, CPA-secure PRE scheme for message domain $\mathbb{G}_T$ according to Definition 2.3.*

The main difficulty in the proof of Theorem 3.1 is ensuring that the reduction can properly answer *all* the re-encryption key and re-encryption queries asked by $\mathcal{A}$. Here, it will be more intuitive to work with the following assumption *implied* by EDBDH:

Definition 3.2 Modified Extended Decisional Bilinear Diffie-Hellman (mEDBDH): *Let* $\mathsf{BSetup}(1^k) \rightarrow (q, g, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$*, where* $\langle g \rangle = \mathbb{G}$*. For all p.p.t. adversaries* $\mathcal{A}$*, there exists a negligible function* ε *such that the following probability is less than or equal to 1/2 +* $\varepsilon(k)$*:*

$$\Pr[s,t,u,v \leftarrow \mathbb{Z}_q;\ x_1 \leftarrow \mathbf{e}(g,g)^{st/u};\ x_0 \leftarrow \mathbf{e}(g,g)^{v};\ z \leftarrow \{0,1\};\ z' \leftarrow \mathcal{A}(g, g^s, g^t, g^u, \mathbf{e}(g,g)^{t/u}, x_z) : z = z'].$$

Lemma 3.3 *If the EDBDH assumption holds in* $\mathbb{G}$*, then so does the mEDBDH assumption.*

Proof. Suppose adversary $\mathcal{B}$ can decide mEDBDH, we can construct $\mathcal{B}'$ that solves the EDBDH problem. On input $(g, g^a, g^b, g^c, \mathbf{e}(g,g)^{bc^2}, Q)$, $\mathcal{B}'$ sends $(g^c, g^a, g^b, g, \mathbf{e}(g,g)^{bc^2}, Q)$ to $\mathcal{B}$ and returns the same answer. To see this is a perfect simulation let $y = g^c$. Then $g^a = y^{a/c} = y^{\alpha}$, similarly $g^b = y^{b/c} = y^{\beta}$ and $g = y^{1/c} = y^{\gamma}$. The value of $\mathbf{e}(g,g)^{bc^2} = \mathbf{e}(y,y)^{b} = \mathbf{e}(y,y)^{(b/c)/(1/c)} = \mathbf{e}(y,y)^{\beta/\gamma}$. Similarly, the value of $\mathbf{e}(g,g)^{abc} = \mathbf{e}(y,y)^{ab/c} = \mathbf{e}(y,y)^{(a/c)(b/c)/(1/c)} = \mathbf{e}(y,y)^{\alpha\beta/\gamma}$. □

We now proceed with the proof of Theorem 3.1.

Proof. Suppose $\mathcal{A}$ breaks the CPA-security of our PRE construction with probability $1/2 + \mu$, then we create an adversary $\mathcal{B}$ who breaks the mEDBDH assumption with probability $1/2 + \mu/2$. Recall that mEDBDH asks when given $(g,\ g^s,\ g^t,\ g^u,\ \mathbf{e}(g,g)^{t/u},\ Q)$ is $Q = \mathbf{e}(g,g)^{st/u}$. Given a mEDBDH instance Δ, $\mathcal{B}$ handles oracle queries from $\mathcal{A}$ as:

- **Public Parameter Generation** $\mathcal{B}$ sets up the global parameters for $\mathcal{A}$ by selecting a random $n \in \mathbb{Z}_q$ and setting $(g, h, Z) = (g, g^n, \mathbf{e}(g,g)^n)$.
- **Uncorrupted Key Generation** $\mathcal{B}$ chooses random $x, y \in \mathbb{Z}_q$ and outputs the public key $pk = ((\mathbf{e}(g,g)^{t/u})^{nx}, (g^u)^y)$, where the secret key is implicitly defined as $sk = (tx/u, uy)$.
- **Corrupted Key Generation** $\mathcal{B}$ choose random $x_i, y_i \in \mathbb{Z}_q$, and outputs $pk_i = (Z^{x_i}, g^{y_i})$ and $sk_i = (x_i, y_i)$.
- **Re-encryption Key Generation** On input (i, j) to $\mathcal{O}_{rkey}$, output the following:
  - If (1) i is uncorrupted and j is corrupted or (2) $i = j$, output $\perp$.
  - If i and j are corrupted, pick random $r, w \in \mathbb{Z}_q$ and output $(g^{y_j(x_i+r)}, h^r, \mathbf{e}(g^{y_j}, h)^w,\ Z^w)$.
  - If i and j are both uncorrupted, select a random $r, w \in \mathbb{Z}_q$ and output $((g^t)^{y_j x_i} \cdot (g^u)^{y_j r}, h^r, \mathbf{e}((g^u)^{y_j}, h)^w, Z^w)$. The first term of the re-encryption key from Alice to Bob looks like $(g^{b_2})^{a_1+r} = (g^{uy_b})^{tx_a/u+r} = g^{tx_a y_b} \cdot g^{uy_b r}$.
  - If i is corrupted and j is uncorrupted, select a random $r, w \in \mathbb{Z}_q$ and output the key $((g^u)^{y_j(x_i+r)}, h^r, \mathbf{e}((g^u)^{y_j}, h)^w, Z^w)$.
- **Re-encryption** On input $(i, j, C_i = (\alpha, \beta, \gamma))$ to $\mathcal{O}_{renc}$, output the following:
  - If (1) i is uncorrupted and j is corrupted, (2) $i = j$ or (3) $\mathbf{e}(\alpha, h) \neq \mathbf{e}(g, \beta)$, output $\perp$.
  - Otherwise, there exists some $k \in \mathbb{Z}_q$ and $m \in \mathbb{G}_T$ such that $\alpha = g^k$, $\beta = h^k$ and $\gamma = m \cdot Z^{tx_i k/u}$ if i is honest or $\gamma = m \cdot Z^{kx_i}$ if i is corrupted.
  - If i and j are both corrupted, recover $m = \gamma / \mathbf{e}(g, \beta)^{x_i}$, then select a random $r \in \mathbb{Z}_q$ and output $(Z^{y_j r}, m \cdot Z^r)$.
  - If i and j are both uncorrupted, select random $r, w \in \mathbb{Z}_q$ and output $(\mathbf{e}(g^{ty_j x_i} \cdot g^{uy_j r}, \beta) \cdot \mathbf{e}(g^u, h)^{y_j w}, \gamma \cdot \mathbf{e}(\alpha, h^r) \cdot Z^w) = (\mathbf{e}(g^{ty_j x_i} \cdot g^{uy_j r}, h^k) \cdot \mathbf{e}(g^u, h)^{y_j w}, m \cdot Z^{kx_i t/u} \cdot \mathbf{e}(g^k, h^r) \cdot Z^w)$.
  - If i is corrupted and j is uncorrupted, recover $m = \gamma / \mathbf{e}(g, \beta)^{x_i}$, then select a random $r \in \mathbb{Z}_q$ and output $(\mathbf{e}(g^u, h)^{y_j r}, m \cdot Z^r)$.

- **Challenge Oracle** Challenges are of the form (i, m_0, m_1) where i is the index of an uncorrupted user. $\mathcal{B}$ responds by choosing random $d \in \{0, 1\}$ and outputting the ciphertext: $(g^s, (g^s)^n, m_d \cdot Q^{nx_i})$.
- **Decision** $\mathcal{A}$ will submit a guess of $d' \in \{0, 1\}$. If $d = d'$ then $\mathcal{B}$ outputs 1 (is a mEDBDH instance) otherwise it outputs 0 (not a mEDBDH instance).

By construction, the public parameters and all uncorrupted keys, corrupted keys, re-encryption keys, and re-encryptions are correct and distributed properly. As to the challenge ciphertext, we have two cases. In the case that $Q = \mathbf{e}(g, g)^{st/u}$, then the challenge ciphertext is a proper encryption of m_d. $\mathcal{A}$ will output d' such that $d = d'$ with probability $\frac{1}{2} + \mu$. Consequently, $\mathcal{B}$ will determine that Δ was a mEDBDH instance and answer 1 with the same probability. When Q is random, independent of s, t and u then the challenge ciphertext reveals no information about m_d. $\mathcal{A}$ will guess that $d = d'$ with probability of exactly $\frac{1}{2}$, and $\mathcal{B}$ will correctly output 0 (not a mEDBDH) with the same probability. The probability that Δ is a valid mEDBDH instance is $\frac{1}{2}$, and $\mathcal{B}$ will output the correct answer with probability: $(\frac{1}{2})(\frac{1}{2} + \mu) + (\frac{1}{2})(\frac{1}{2}) = \frac{1}{2} + \frac{\mu}{2}$. We apply Lemma 3.3 to establish the result. □

Theorem 3.4 (Key Privacy) *Under the Decision Linear assumption in* $\mathbb{G}$*, scheme* Π *is a unidirectional, single-hop, key-private PRE scheme according to Definition 2.6.*

The key-privacy proof is more difficult than that of CPA security. In particular, here we must be able to correctly re-encrypt ciphertexts for a special pair of users (I, J) even though we may not be able to compute a valid re-encryption key from I to J. To do this, we designed our encryption scheme in such a way that there is a "back door" for decryption, which in some cases, allows us to decrypt and then encrypt (thus simulating re-encryption) even when we cannot directly compute the re-encryption key needed to run the real re-encryption algorithm.

Proof. We show that if an adversary $\mathcal{A}$ can break the key-privacy game with probability $1/2+\mu$, then we can construct an adversary $\mathcal{B}$ who can break the Decision Linear assumption with probability roughly $\frac{1}{2} + \frac{\mu}{4n^2}$, where n is the number of honest users. (This loose bound comes from letting the adversary dynamically pick its pair of honest users to challenge on. In prior key-privacy definitions [3], the adversary was restricted to a single pair.)

Given a Decision Linear input $\Delta' = (g, h, f, g^x, h^y, Q \stackrel{?}{=} f^{x+y})$, $\mathcal{B}$ handles oracle queries from $\mathcal{A}$ as follows. Let n be the bound on the number of uncorrupted users which $\mathcal{A}$ will ask to be created. $\mathcal{B}$ randomly chooses two as special users I and J, out of these n, and predicts that $\mathcal{A}$ will challenge on identities (I, J). $\mathcal{B}$ will proceed to set up things, so that first two elements of the valid re-encryption key from I to J will be (f^{x+y}, h^y). At a high-level, if $\mathcal{A}$ challenged on (I, J) then his response will be used to help $\mathcal{B}$, and if $\mathcal{A}$ challenges on anything else $\mathcal{B}$ will abort. Fortunately, we will see that $\mathcal{B}$ does not abort with probability $\geq 1/2n^2$.

Assuming (I, J) are chosen, let's see how $\mathcal{B}$ proceeds:

- **Public Parameter Generation** $\mathcal{B}$ sets up the parameters of $\mathcal{A}$ as $(g, h, Z) = (g, h, \mathbf{e}(g, h))$.
- **Uncorrupted Key Generation**
  - If this is the key for special user I, then select random $a \in \mathbb{Z}_q$ and output $(\mathbf{e}(g^x, h), g^a)$.
  - If this is the key for special user J, then select random $b \in \mathbb{Z}_q$ output (Z^b, f). Denote $f := g^s$, for some $s \in \mathbb{Z}_q$.

– Otherwise select random $m_i, n_i \in \mathbb{Z}_q$ and output (Z^{m_i}, g^{n_i}).

- **Corrupted Key Generation** Select random $m_i, n_i \in \mathbb{Z}_q$ and output the public key as (Z^{m_i}, g^{n_i}), as well as the private key pair (m_i, n_i).
- **Re-encryption Key Generation** Given a request to encrypt from i to j, $\mathcal{B}$ selects a random $r \in \mathbb{Z}_q$ and proceeds as follows. If i is corrupted, this computation can be done by $\mathcal{A}$.
  - If $\mathcal{B}$ produced a re-encryption key from i to j before or $i = j$, output $\perp$.
  - If i is I and j is J, then abort. (This means (I, J) will not be the challenge pair.)
  - If i is I and j is other, then output $((g^x)^{n_j} \cdot g^{n_j r}, h^r, Z^{n_j w}, Z^w)$.
  - If i is J and j is I, then output $(g^{a(b+r)}, h^r, Z^{aw}, Z^w)$.
  - If i is J and j is other, then output $((g^{n_j})^{b+r}, h^r, Z^{n_j w}, Z^w)$.
  - If i is other and j is I, then output $(g^{a(m_i+r)}, h^r, Z^{aw}, Z^w)$.
  - If i is other and j is J, then output $(f^{m_i+r}, h^r, \mathbf{e}(f,h)^w, Z^w)$.
  - If i is other and j is other, then output $(g^{n_j(m_i+r)}, h^r, Z^{n_j w}, Z^w)$.
- **Re-encryption** On input (i, j, C_i),
  - Check that $C_i = (\alpha, \beta, \gamma)$ is properly formed by testing if $\mathbf{e}(\alpha, h) = \mathbf{e}(g, \beta)$. If this check fails or $i = j$, output $\perp$.
  - If (i, j) is (I, J), then decrypt C_i using g^x as $m = \gamma / \mathbf{e}(g^x, \beta)$. Choose a random $r \in \mathbb{Z}_q$ and output $(\mathbf{e}(f,h)^r, m \cdot Z^r) = (Z^{sr}, m \cdot Z^r)$. This is a key step in the proof.
  - Otherwise obtain a re-encryption key $(\zeta, \eta, \theta, \lambda)$ of the same form as the re-encryption oracle. (It will not matter here if the same key is used or a new key generated each time, since the next step re-randomizes the output to hide which key was used.) The ciphertext is then re-encrypted and re-randomized by selecting a random $w' \in \mathbb{Z}_q$ and the output is $(\mathbf{e}(\zeta, \beta) \cdot \theta^{w'}, \gamma \cdot \mathbf{e}(\alpha, \eta) \cdot \lambda^{w'})$.
- **Challenge Oracle** Challenges are of the form (i, j) where i and j are indices of uncorrupted users which have not been queried before. If (i, j) is (I, J), $\mathcal{B}$ outputs $(Q, h^y, e(f,h)^w, Z^w)$. Otherwise, $\mathcal{B}$ aborts and makes a random guess.
- **Decision** $\mathcal{A}$ will submit a guess of $d \in \{0, 1\}$. If $d = 1$, then $\mathcal{B}$ outputs 1 (is a Decision Linear instance), otherwise it outputs 0 (not a Decision Linear instance).

The public parameters, key generation algorithm, and all responses of $\mathcal{O}_{rkey}$ and $\mathcal{O}_{renc}$ are correct and properly distributed. When $\mathcal{B}$ does not abort on the challenge and $\mathcal{A}$ does not detect improper queries, we have two cases. If $Q = f^{x+y}$, then the challenge is a valid re-encryption key for (I, J) and $\mathcal{A}$ will output 1 (a good re-encryption key) with probability $\frac{1}{2} + \mu$. $\mathcal{B}$ will output the correct answer (is Decision Linear instance) with the same probability. If Q is random, then $\mathcal{A}$ will output the correct answer of 0 (not a valid re-encryption key) with probability $\frac{1}{2}$. $\mathcal{B}$ outputs the same answer, so it will correctly determine that Δ' is not a Decision Linear instance with the same probability. Given that each case occurs with probability 1/2 when $\mathcal{B}$ does not abort and $\mathcal{B}$ does not abort with probability $\geq \frac{1}{2n^2}$, the total probability of $\mathcal{B}$'s success is $\geq \frac{1}{2} + \frac{1}{2n^2} \cdot \frac{\mu}{2}$. □

Remark 3.5 (On Multiple Keys per Pair) The definition of key privacy in Section 2 (as well as the obfuscation definition of [9]) allows only one key between any pairs of users to be given out, even for probabilistic schemes where there may be many possible keys. For completeness, we mention the scheme in Section 3.2 does not remain key-private if multiple keys per pair are released.

Consider the following attack: given two keys with different randomness from A to B:

$$\begin{array}{ccc} g^{b_2(a_1+r_1)} & , & h^{r_1} \\ g^{b_2(a_1+r_2)} & , & h^{r_2} \end{array}$$

the adversary can discover if the delegatee is g^{b_2}, by computing $J_1 = g^{b_2(r_1-r_2)}$ and $J_2 = h^{(r_1-r_2)}$, and then testing if $\mathbf{e}(J_1, h) = \mathbf{e}(b^{b_2}, J_2)$. It would be interesting if a stronger definition could be realized.

4 Conclusions and Open Problems

We formalized the notion of *key-privacy* for proxy re-encryption schemes, where even the proxy performing the translations cannot distinguish the identities of the participating parties. We discussed why none of the six or more existing PRE schemes satisfy this simple privacy notion. We then presented the first construction under standard assumptions in the standard model. Our construction is efficient enough for several practical PRE applications, such as [1, 13, 10, 12].

Our construction realizes CPA-security. It would be interesting to realize key-private CCA-secure PRE. However, some basic approaches, such as applying the CPA-to-CCA transformation of Fujisaki and Okamoto [8] do not appear to maintain the key-privacy properties. It was also surprising that the definition of obfuscation, as in [9], does not capture key-privacy. It would be very interesting to know if a secure obfuscation of PRE could be realized when allowing the proxy and users to collude and allowing all the re-encryption and re-encryption key queries admitted here, as they would be in a real system. Unfortunately, these types of collusions and interactions appear to break the *security* of the construction of [9] even without considering key-privacy.

Finally, we suspect that simpler key-private PRE schemes can be devised, although at the cost of stronger assumptions. The extended version of DBDH and the Decision Linear assumptions used here are actually quite mild. Nevertheless, finding more efficient schemes, even under stronger assumptions or in the random oracle model, would be quite useful for several applications.

Acknowledgments

The authors thank Jun Shao and the anonymous reviewers for helpful comments. The authors gratefully acknowledge the support of NSF grant CNS-0716142. Susan Hohenberger also acknowledges support from a Microsoft New Faculty Fellowship.

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 and Susan Hohenberger. Proxy re-signatures: new definitions, algorithms, and applications. In *ACM Conference on Computer and Communications Security*, pages 310–319, 2005.

[3] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In *ASIACRYPT*, pages 566–582, 2001.

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

[5] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, pages 41–55, 2004.

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

[7] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *ACM CCS*, pages 185–194. ACM, 2007.

[8] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology CRYPTO 99*, volume 1666, page 79, 1999.

[9] 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.

[10] Himanshu Khurana, Jin Heo, and Meenal Pant. From proxy encryption primitives to a deployable secure-mailing-list solution. In *ICICS*, pages 260–281, 2006.

[11] Benoit Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In *PKC '08*, volume 4939 of LNCS, pages 360–379, 2008.

[12] Suriadi Suriadi, Ernest Foo, and Jason Smith. Conditional privacy using re-encryption. In *IFIP International Workshop on Network and System Security*, 2008.

[13] Gelareh Taban, Alvaro A. Cárdenas, and Virgil D. Gligor. Towards a secure and interoperable DRM architecture. In *DRM '06: Proceedings of the ACM workshop on Digital rights management*, pages 69–78. ACM, 2006.

A Proof of Theorem A.1

Theorem A.1 *If the DBDH assumption holds in $\mathbb{G}$, then the first-level ciphertext of the Section 3.2 scheme is CPA-secure.*

Proof. Suppose $\mathcal{A}$ breaks the CPA-security of our PRE construction with probability $1/2+\mu$, then we create an adversary $\mathcal{B}$ who breaks the DBDH assumption with probability $1/2+\mu/2$. Recall that DBDH asks when given (g, g^a, g^b, g^c, Q), where $a, b, c \in \mathbb{Z}_q$, is $Q = \mathbf{e}(g,g)^{abc}$.

Given a DBDH instance Δ, $\mathcal{B}$ handles oracle queries from $\mathcal{A}$ as:

- **Public Parameter Generation** $\mathcal{B}$ sets up the global parameters of $\mathcal{A}$ as $(g, h, Z) = (g, g^c, \mathbf{e}(g, g^c))$.
- **Uncorrupted Key Generation** $\mathcal{B}$ chooses random $x_i, y_i \in \mathbb{Z}_q$, and outputs pk_i=(Z^{x_i}, $(g^a)^{y_i}$).

- **Corrupted Key Generation** $\mathcal{B}$ chooses random $x_i, y_i \in \mathbb{Z}_q$, and outputs pk_i=(Z^{x_i}, g^{y_i}) and sk_i=(x_i, y_i).
- **Re-encryption Key Generation** On input (i, j) to $\mathcal{O}_{rkey}$ check if i and j are corrupted and output the following:
  - If i is uncorrupted and j is corrupted, or $i = j$, output $\perp$.
  - If i and j are both corrupted, pick random $r, w \in \mathbb{Z}_q$ and output ($g^{y_j(x_i+r)}$, h^r, $\mathbf{e}(g^{y_j}, h)^w$, Z^w).
  - Otherwise select a random $r, w \in \mathbb{Z}_q$ and output ((g^a)$^{y_j(x_i+r)}$, h^r, $\mathbf{e}(g^{ay_j}, h)^w$, Z^w).
- **Re-encryption** On input $(i, j, C_i = (\alpha, \beta, \gamma))$ to $\mathcal{O}_{renc}$ check if i and j are corrupt and output the following:
  - If (1) i is uncorrupted and j is corrupted, (2) $i = j$ or (3) $\mathbf{e}(\alpha, h) \neq \mathbf{e}(g, \beta)$, output $\perp$.
  - Otherwise, there exists some $k \in \mathbb{Z}_q$ and $m \in \mathbb{G}_T$ such that $\alpha = g^k$, $\beta = h^k$ and $\gamma = m \cdot Z^{x_i k}$.
  - If i and j are both corrupted, recover $m = \gamma / \mathbf{e}(g, \beta)^{x_i}$, then select a random $r \in \mathbb{Z}_q$ and output $(Z^{y_j r}, m \cdot Z^r)$.
  - In all other cases, recover $m = \gamma / \mathbf{e}(g, \beta)^{x_i}$, then select a random $r \in \mathbb{Z}_q$ and output $(\mathbf{e}(g^a, h)^{y_j r}, m \cdot Z^r) = (Z^{a y_j r}, m \cdot Z^r)$.
- **Challenge Oracle** Challenges are of the form (i, m_0, m_1) where i is the index of an uncorrupted user. $\mathcal{B}$ responds by choosing random $d \in \{0, 1\}$ and outputting the first-level ciphertext: $(Q^{y_i}, m_d \cdot \mathbf{e}(g^b, h))$.
- **Decision** When $\mathcal{A}$ submits a guess of $d' \in \{0, 1\}$. If $d = d'$ then $\mathcal{B}$ outputs a 1 meaning Δ is a DBDH tuple, otherwise 0 meaning Δ is not a DBDH tuple.

By construction, the public parameters and all uncorrupted keys, corrupted keys, re-encryption keys, and re-encryptions are correct and distributed properly. As to the challenge ciphertext, we have two cases. In the case that $Q = \mathbf{e}(g, g)^{abc}$, then the challenge ciphertext is (Z^{aby_i}, $m_d \cdot Z^b$) whichis a proper encryption of m_d. $\mathcal{A}$ will output d' such that $d = d'$ with probability $\frac{1}{2} + \mu$. Consequently, $\mathcal{B}$ will determine that Δ was a DBDH instance and answer 1 with the same probability. When Q is random, independent of a, b and c then the challenge ciphertext reveals no information about m_d. $\mathcal{A}$ will guess that $d = d'$ with probability of exactly $\frac{1}{2}$, and $\mathcal{B}$ will correctly output 0 (not a DBDH) with the same probability. The probability that Δ is a valid DBDH instance is $\frac{1}{2}$, and $\mathcal{B}$ will output the correct answer with probability: $(\frac{1}{2})(\frac{1}{2} + \mu) + (\frac{1}{2})(\frac{1}{2}) = \frac{1}{2} + \frac{\mu}{2}$. □