Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage

Giuseppe Ateniese* Kevin Fu† Matthew Green* Susan Hohenberger†

Abstract

In 1998, Blaze, Bleumer, and Strauss proposed an application called atomic proxy re-encryption*, in which a semi-trusted proxy converts a ciphertext for Alice into a ciphertext for Bob* without *seeing the underlying plaintext. We predict that fast and secure re-encryption will become increasingly popular as a method for managing encrypted file systems. Although efficiently computable, the wide-spread adoption of BBS re-encryption has been hindered by considerable security risks. Following recent work of Ivan and Dodis, we present new re-encryption schemes that realize a stronger notion of security and we demonstrate the usefulness of proxy re-encryption as a method of adding access control to the SFS read-only file system. Performance measurements of our experimental file system demonstrate that proxy re-encryption can work effectively in practice.*

1. Introduction

Proxy re-encryption allows a proxy to transform a ciphertext computed under Alice's public key into one that can be opened by Bob's secret key. There are many useful applications of this primitive. For instance, Alice might wish to temporarily forward encrypted email to her colleague Bob, without giving him her secret key. In this case, Alice the delegator could designate a proxy to re-encrypt her incoming mail into a format that Bob the delegatee can decrypt using his own secret key. Clearly, Alice could provide her secret key to the proxy but this requires an unrealistic level of trust in the proxy.

We present several efficient proxy re-encryption schemes that offer security improvements over earlier approaches. The primary advantage of our schemes is that they are unidirectional (i.e., Alice can delegate to Bob without Bob having to delegate to her) and do not require delegators to reveal all of their secret key to anyone– or even interact with the delegatee– in order to allow a proxy to re-encrypt their ciphertexts. In our schemes, only a limited amount of trust is placed in the proxy. For example, it is not able to decrypt the ciphertexts it re-encrypts and we prove our schemes secure even when the proxy publishes all the re-encryption information it knows. This enables a number of applications that would not be practical if the proxy needed to be fully trusted.

We present an application for proxy cryptography in securing distributed file systems. Our system uses a centralized *access control server* to manage access to encrypted files stored on distributed, untrusted replicas. We use proxy re-encryption to allow for centrally-managed access control without granting full decryption rights to the access control server.

No experimental implementation of proxy re-encryption schemes has been provided, to our knowledge, which makes it difficult to argue about the effectiveness of the proxy re-encryption primitive. In this paper, we provide new protocols with improved security guarantees (based on bilinear maps) and demonstrate their practicality based on runtime experiments.

1.1. Proxy Re-encryption Background

A methodology for delegating decryption rights was first introduced by Mambo and Okamoto [30] purely as an efficiency improvement over traditional decrypt-and-then-encrypt approaches.

In 1998, Blaze, Bleumer, and Strauss [6] proposed the notion of "atomic proxy cryptography", in which a semi-trusted proxy computes a function that converts ciphertexts for Alice into ciphertexts for Bob without see-

*Department of Computer Science; The Johns Hopkins University; 3400 N. Charles Street; Baltimore, MD 21218, USA. Email: {ateniese,mgreen}@cs.jhu.edu.

†Computer Science and Artificial Intelligence Laboratory; Massachusetts Institute of Technology; 32 Vassar Street; Cambridge, MA 02139, USA. Email: {fubob,srhohen}@mit.edu. Some of the research of S. Hohenberger and K. Fu was performed while visiting IBM Research Labs, Zurich, Switzerland and the Johns Hopkins University Information Security Institute respectively.

ing the underlying plaintext. In their El Gamal based scheme, with modulus a safe prime $p = 2q+1$, the proxy is entrusted with the delegation key $b/a \bmod q$ for the purpose of diverting ciphertexts from Alice to Bob via computing $(mg^k \bmod p, (g^{ak})^{b/a} \bmod p)$. The authors noted, however, that this scheme contained an inherent restriction: it is *bidirectional*; that is, the value b/a can be used to divert ciphertexts from Alice to Bob and vice versa. Thus, this scheme is only useful when the trust relationship between Alice and Bob is mutual. (This problem can be solved, for any scheme, by generating an additional, otherwise unused, key pair for the delegatee, but this introduces additional overhead.) The BBS scheme contains further problems. Delegation in the BBS scheme is *transitive*, which means that the proxy alone can create delegation rights between two entities that have never agreed on this. For example, from the values a/b and b/c, the proxy can re-encrypt messages from Alice to Carol. Another drawback to this scheme is that if the proxy and Bob collude, they can recover her secret key as $(a/b) * b = a$!

Jakobsson [26] developed a quorum-based protocol where the proxy is divided into sub-components, each controlling a share of the re-encryption key; here, the keys of the delegator are safe so long as some of the proxies are honest. A similar approach was considered by Zhou, Mars, Schneider and Redz [39].

Recently, Ivan and Dodis [14] realized *unidirectional* proxy encryption for El Gamal, RSA, and an IBE scheme by sharing the user's secret key between two parties. They also solved the problem of the proxy alone assigning new delegation rights. In their unidirectional El Gamal scheme, Alice's secret key s is divided into two shares s_1 and s_2, where $s = s_1 + s_2$, and distributed to the proxy and Bob. On receiving ciphertexts of the form (mg^{sk}, g^k), the proxy first computes $(mg^{sk}/(g^k)^{s_1})$, which Bob can decrypt as $(mg^{s_2 k}/(g^k)^{s_2}) = m$. Although this scheme offers some advantages over the BBS approach, it introduces new drawbacks as well. These "secret-sharing" schemes do not change ciphertexts for Alice into ciphertexts for Bob in the purest sense (i.e., so that Bob can decrypt them with *his* own secret key), they delegate decryption by requiring Bob to store additional secrets (i.e., shares $\{s_2^{(i)}\}$) that may in practice be difficult for him to manage. For example, in our file system in Section 4, the number of secrets a user must manage should remain constant regardless of the number of files it accesses. One exception is the Ivan-Dodis IBE scheme [14] where the global secret that decrypts *all* ciphertexts is shared between the proxy and the delegatee. Thus, the delegatee need only store a single secret, but an obvious drawback is that when the proxy and any delegatee in the system collude, they can decrypt everyone else's messages.

Thus, proxy re-encryption protocols combining the various advantages of the BBS and Ivan-Dodis schemes, along with new features such as time-limited delegations, remained an open problem. (We provide a list of these desirable features in Section 3.) Our results can be viewed as contributing both to the set of key-insulated [12, 13, 15] and signcryption [3, 4, 38] schemes, where Alice may expose her secret key without needing to change her public key and/or use the same public key for encryption and signing purposes. This work should not be confused with the "universal re-encryption" literature [24], which re-randomizes ciphertexts instead of changing the public key.

1.2. Applications of Proxy Re-encryption

Proxy re-encryption has many exciting applications in addition to the previous proposals [6, 14, 26, 39] for email forwarding, law enforcement, and performing cryptographic operations on storage-limited devices. In particular, proxy cryptography has natural applications to secure network file storage. The following paragraphs describe potential applications of proxy re-encryption.

Secure File Systems. A secure file system is a natural application of proxy re-encryption because the system often assumes a model of untrusted storage.

A number of distributed file systems build confidential storage out of untrusted components by using cryptographic storage [2, 5, 22, 28]. Confidentiality is obtained by encrypting the contents of stored files. These encrypted files can then be stored on untrusted file servers. The server operators can distribute encrypted files without having access to the plaintext files themselves.

In a single-user cryptographic file system, access control is straightforward. The user creates all the keys protecting content. Thus, there is no key distribution problem. With group sharing in cryptographic storage, group members must rendezvous with content owners to obtain decryption keys for accessing files.

Systems with cryptographic storage such as the SWALLOW object store [32] or CNFS [25] assume an out-of-band mechanism for distributing keys for access control. Other systems such as Cepheus [18] use a trusted access control server to distribute keys.

The access control server model requires a great deal of trust in the server operator. Should the operator prove unworthy of this trust, he or she could abuse the server's key material to decrypt any data stored on the system.

Furthermore, even if the access control server operator is trustworthy, placing so much critical key data in a single location makes for an inviting target.

In contrast, our system makes use of a semi-trusted access control server. We propose a significant security improvement to the access control in cryptographic storage, using proxy cryptography to reduce the amount of trust in the access control server. In our approach, keys protecting files are stored encrypted under a master public key, using one of the schemes in Section 3. When a user requests a key, the access control server uses proxy cryptography to directly re-encrypt the appropriate key to the user without learning the key in the process. Because the access control server does not itself possess the master secret, it cannot decrypt the keys it stores. The master secret key can be stored offline, by a content owner who uses it only to generate the re-encryption keys used by the access control server. In Section 4, we describe our implementation and provide a performance evaluation of our constructions.

Outsourced Filtering of Encrypted Spam. Another promising application of proxy re-encryption is the filtering of encrypted emails performed by authorized contractors. The sheer volume of unsolicited email, along with rapid advances in filter-avoidance techniques, has overwhelmed the filtering capability of many small businesses, leading to a potential market for *outsourced email filtering*. New privacy regulations, such as the US Health Insurance Portability and Accountability Act (HIPAA), are encouraging companies to adopt institution-wide email encryption to ensure confidentiality of patient information [1]. By accepting encrypted email from outside sources, institutions become "spam" targets and filters are only effective on messages that are first decrypted (which could be unacceptably costly). Using proxy re-encryption, it becomes possible to redirect incoming encrypted email to an external filtering contractor at the initial mail gateway, without risking exposure of plaintexts at the gateway itself. Using our *temporary* proxy re-encryption scheme presented in Section 3.2, a healthcare institution can periodically change filtering contractors without changing its public key.

1.3. Roadmap

The rest of this paper consists of the following. Section 2 gives some number theoretic preliminaries and definitions necessary to understand our schemes and their security guarantees. Section 3 presents improved proxy re-encryption schemes as well as a discussion on the factors to consider when comparing proxy re-encryption schemes. Section 4 highlights the design, implementation, and performance measurements of our proxy re-encryption file system. We provide concluding remarks in Section 5.

2. Definitions

Our protocols are based on bilinear maps [7, 8, 9, 27]), which we implemented using the fast Tate pairings [21].

Definition 2.1 (Bilinear Map) We say a map $e : G_1 \times \hat{G}_1 \rightarrow G_2$ is a *bilinear map* if: (1) G_1 and G_2 are groups of the same prime order q; (2) for all $a, b \in \mathbb{Z}_q$, $g \in G_1$, and $h \in \hat{G}_1$, then $e(g^a, h^b) = e(g, h)^{ab}$ is efficiently computable; (3) the map is non-degenerate (i.e., if g generates G_1 and h generates $\hat{G}_1$, then $e(g, h)$ generates G_2); and (4) there exists a computable isomorphism from $\hat{G}_1$ to G_1. (Here, G_1 may equal $\hat{G}_1$.)

Now, we explain the different components of a proxy re-encryption scheme. We will be informal in this section, referring an interested reader to Appendix A for precise definitions.

Definition 2.2 (Unidirectional Proxy Re-encryption) *A* re-encryption scheme *is a tuple of (possibly probabilistic) polynomial time algorithms* $(KG, RG, \vec{E}, R, \vec{D})$*, where:*

- $(KG, \vec{E}, \vec{D})$ form the standard key generation, encryption, and decryption algorithms.
- On input $(pk_A, sk_A, pk_B, sk_B^*)$, the re-encryption key generation algorithm, RG, outputs a key $rk_{A\rightarrow B}$ for the proxy. The fourth input marked with a '$*$' is optional.
- On input $rk_{A\rightarrow B}$ and ciphertext C_A, the re-encryption function, R, outputs C_B.

Correctness. Alice should always be able to decrypt ciphertexts encrypted under pk_A; while Bob should be able to decrypt re-encrypted ciphertexts $R(rk_{A\rightarrow B}, C_A)$. The encryption algorithm $\vec{E}$ may allow multiple types of encryption, such as *first-level* encryptions that cannot be re-encrypted and *second-level* encryptions that can be. This gives the sender a choice between encrypting a message only to Alice or to Alice and her delegatee Bob within the same system. For re-encrypted ciphertexts, we require that the underlying plaintext remain *consistent* – i.e., Bob should get exactly what Alice was supposed to receive.[1]

[1]Note, this only applies to ciphertexts that were honestly generated by the sender; no guarantee is implied in the case of malformed ciphertexts.

Security. Our security definition has two parts: *standard security*, requiring that the cryptosystem remain semantically-secure [23] even when all re-encryption keys are public; and *master secret security*, where a delegator's master secret key is not recoverable. Our security definition is similar to that of Ivan and Dodis [14]. Although their definition was for CCA2 security, they instead used CPA security for the El Gamal, RSA, and IBE-based schemes. The first main difference between our definitions is that we consider the security of a user against a *group* of colluding parties; i.e., the security of a delegator against the proxy and many delegatees, whereas the Ivan-Dodis definition focused on a single delegatee. Second, we discuss the system's security for circular delegations; that is, when an adversary watches Alice and Bob delegate to each other. Finally, master secret security is a new guarantee for the delegator.

Property	BBS [6]	ID [14]	This work
1. Unidirectional	No	Yes	Yes
2. Non-interactive	No	Yes	Yes
3. Proxy invisible	Yes	No	Yes
4. Original-access	Yes†	Yes	Yes†
5. Key optimal	Yes	No	Yes
6. Collusion-"safe"	No	No	Yes*
7. Temporary	Yes†	Yes†	Yes†
8. Non-transitive	No	Yes	Yes
9. Non-transferable	No	No	No

Table 1. We compare known proxy re-encryption schemes based on the advantages described above; no scheme achieves property 9. We refer to the unidirectional schemes of Ivan-Dodis. $*$ indicates master secret key only. † indicates possible to achieve with additional overhead.

3. Improved Proxy Re-encryption Schemes

To talk about "improvements", we need to get a sense of the benefits and drawbacks of previous schemes. Here is a list of, in our opinion, the most useful properties of proxy re-encryption protocols:

1. *Unidirectional:* Delegation from $A \rightarrow B$ does not allow re-encryption from $B \rightarrow A$.

2. *Non-interactive:* Re-encryption keys can be generated by Alice using Bob's public key; no trusted third party or interaction is required. (Such schemes were called *passive* in BBS [6].)

3. *Proxy invisibility:* This is an important feature offered by the original BBS scheme. The proxy in the BBS scheme is *transparent* in the sense that neither the sender of an encrypted message nor any of the delegatees have to be aware of the existence of the proxy. Clearly, transparency is very desirable but it is achieved in the BBS scheme at the price of allowing transitivity of delegations and recovery of the master secrets of the participants. Our pairing-based schemes, to be described shortly, offer a weaker form of transparency which we call *proxy invisibility.* In particular, we allow the sender to be aware of the proxy and decide whether to generate an encryption that can be opened only by the intended recipient (*first-level encryption*) or by any of the recipient's delegatees (*second-level encryption*). On the other hand, we can ensure that any delegatee will not be able to distinguish a first-level encryption (computed under his public key) from a re-encryption of a ciphertext intended for another party (we are assuming that the encrypted message does not reveal information that would help the delegatee to make this distinction).

4. *Original-access:* Alice can decrypt re-encrypted ciphertexts that were originally sent to her. In some applications, it may be desirable to maintain access to her re-encrypted ciphertexts. This is an inherent feature of the Ivan-Dodis schemes (since the re-encryption key is a share of the original); the BBS scheme and the pairing schemes presented here can achieve this feature by adding an additional term to the ciphertext: for example, in BBS a re-encrypted ciphertext with original access looks like $(mg^k, g^{ak}, (g^{ak})^{b/a})$. This may impact proxy invisibility.

5. *Key optimal:* The size of Bob's secret storage remains constant, regardless of how many delegations he accepts. We call this a *key optimal* scheme. In the previous El Gamal and RSA based schemes [14], the storage of both Bob and the proxy grows linearly with the number of delegations Bob accepts. This is an important consideration, since the safeguarding and management of secret keys is often difficult in practice.

6. *Collusion-"safe":* One drawback of all previous schemes is that by colluding Bob and the proxy can recover Alice's secret key: for Ivan-Dodis, $s = s_1 + s_2$; for BBS, $a = (a/b) * b$. We will mitigate this problem – allowing recovery of a "weak" secret key only. In a bilinear map setting, suppose Alice's public key is $e(g, g)^a$ and her secret key is a; then we might allow Bob and the proxy to recover the value g^a, but not a itself. Thus, Alice can delegate decryption rights, while keeping signing rights for the same public key. In practice, a user can always use two public keys for encryption and signatures, but it is theoretically interesting that she doesn't *need* to do so. Prior work on "signcryption" explored this area (e.g., [38, 4, 3]); here we present, what can be viewed

as, the first "signREcryption" scheme (although we will not be formally concerning ourselves with the security of the signatures in this work).

7. *Temporary:* Ivan and Dodis [14] suggested applying generic key-insulation techniques [15, 12, 13] to their constructions to form schemes where Bob is only able to decrypt messages intended for Alice that were authored during some specific time period i. Citing space considerations, they did not present any concrete constructions. In Section 3.2, we provide a bilinear map construction designed specifically for this purpose. In our construction, a trusted server broadcasts a new random number at each time period, which each user can then use to update their delegated secret keys. This is an improvement over using current key-insulated schemes where the trusted server needs to individually interact with each user to help them update their master (and therefore, delegation) secret keys.

8. *Non-transitive:* The proxy, alone, cannot re-delegate decryption rights. For example, from $rk_{a\to b}$ and $rk_{b\to c}$, he cannot produce $rk_{a\to c}$.

9. *Non-transferable:* The proxy and a set of colluding delegatees cannot re-delegate decryption rights. For example, from $rk_{a\to b}$, sk_b, and pk_c, they cannot produce $rk_{a\to c}$.We are not aware of any scheme that has this property, and it is a very desirable one. For instance, a hospital may be held legally responsible for safeguarding the encrypted files of its patients; thus, if it chooses to delegate decryption capabilities to a local pharmacy, it may need some guarantee that this information "goes no further." First, we should ask ourselves: is *transferability* really preventable? The pharmacy can always decrypt and forward the plaintext files to a drug company. However, this approach requires that the pharmacy remain an active, online participant. What we want to prevent is the pharmacy (plus the proxy) providing the drug company with a secret value that it can use offline to decrypt the hospital's ciphertexts. Again, the pharmacy can trivially send its secret key to the drug company. But in doing so, it assumes a security risk that is as potentially injurious to itself as the hospital. Achieving a proxy scheme that is *non-transferable*, in the sense that the only way for Bob to transfer offline decryption capabilities to Carol is to expose his own secret key, seems to be the main open problem left for proxy re-encryption.

3.1. New Unidirectional Proxy Re-encryption Schemes

A First Attempt. As Ivan and Dodis pointed out [14], one method for constructing proxy re-encryption schemes is to create a cryptosystem that has two decryption algorithms with two different secret keys. These cryptosystems typically support a *first-level* secret key that decrypts all ciphertexts encrypted under the corresponding public key, and a *second-level* secret key that may decrypt only ciphertexts of a certain form under the same public key. Notice that there are trivial solutions to the proxy re-encryption problem when Alice is allowed to use two different key pairs, but we are interested in solutions that minimize the number to keys to safeguard and manage while remaining efficient.

In the full version of this paper, we present a variant of the Paillier cryptosystem with first and second-level secret keys proposed by Cramer and Shoup [11]. Alice keeps the first-level key for herself and sends the second-level key to Bob as a delegation, with a proxy added to further regulate which ciphertexts Bob may read. This approach is interesting, because it is based on an assumption other than bilinear maps; and offers the collusion-"safety" (i.e., protection of the first-level secret key) that the schemes of Ivan-Dodis lack. However, like the Ivan-Dodis schemes [14], it is not key optimal.

A Second Attempt. To minimize a user's secret storage and thus become key optimal, we present the BBS [6], El Gamal based [16] scheme operating over two groups G_1, G_2 of prime order q with a bilinear map $e : G_1^2 \to G_2$. The system parameters are random generators $g \in G_1$ and $Z = e(g, g) \in G_2$.

- **Key Generation (KG).** A user A's key pair is of the form $pk_a = g^a$, $sk_a = a$.
- **Re-Encryption Key Generation (RG).** A user A delegates to B by publishing the re-encryption key $rk_{A\to B} = g^{b/a} \in G_1$, computed from B's public key.
- **First-Level Encryption (E_1).** To encrypt a message $m \in G_2$ under pk_a in such a way that it can only be decrypted by the holder of sk_a, output $c = (Z^{ak}, mZ^k)$.
- **Second-Level Encryption (E_2).** To encrypt a message $m \in G_2$ under pk_a in such a way that it can be decrypted by A and her delegatees, output $c = (g^{ak}, mZ^k)$.
- **Re-Encryption (R).** Anyone can change a second-level ciphertext for A into a first-level ciphertext for B with $rk_{A\to B} = g^{b/a}$. From $c_a = (g^{ak}, mZ^k)$, compute $e(g^{ak}, g^{b/a}) = Z^{bk}$ and publish $c_b = (Z^{bk}, mZ^k)$.
- **Decryption (D_1).** To decrypt a first-level ciphertext $c_a = (\alpha, \beta)$ with $sk = a$, compute $m = \beta/\alpha^{1/a}$.

Discussion of Scheme. This scheme is very attractive;

it is unidirectional, non-interactive, proxy *transparent*, collusion-safe, key optimal, and non-transitive. However, its security requires the assumption that a cannot be derived from seeing the a-th root of a polynomial set of random values (which is plausible in a group of prime order), in addition to the assumption that the following problem is hard in (G_1, G_2):

$$\text{Given } (g, g^a, g^b, Q), \text{ for } g \leftarrow G_1,\ a, b \leftarrow \mathbb{Z}_q \\ \text{and } Q \in G_2, \text{ decide if } Q = e(g,g)^{a/b}.$$

There is evidence that the computational *Bilinear Inverse Diffie-Hellman* problem is hard [37], but this decisional version has not been studied enough. Next, we provide a solution which makes fewer (and more standard) assumptions.

A Third Attempt. The global system parameters (g, Z) remain unchanged.

- **Key Generation (KG).** A user A's key pair is of the form $pk_a = (Z^{a_1}, g^{a_2})$ and $sk_a = (a_1, a_2)$. (A user can encrypt, sign, and delegate decryption rights all under Z^{a_1}; if the value g^{a_2} is present, it signifies that the user is willing to accept delegations.)
- **Re-Encryption Key Generation (RG).** A user A delegates to B publishing the re-encryption key $rk_{A\to B} = g^{a_1 b_2} \in G_1$, computed from B's public information.
- **First-Level Encryption (E_1).** To encrypt a message $m \in G_2$ under pk_a in such a way that it can only be decrypted by the holder of sk_a, output $c_{a,1} = (Z^{a_1 k}, mZ^k)$ (to achieve proxy invisibility output $c_{a,1} = (Z^{a_2 k}, mZ^k)$ at no security loss).
- **Second-Level Encryption (E_2).** To encrypt a message $m \in G_2$ under pk_a in such a way that it can be decrypted by A and her delegatees, output $c_{a,r} = (g^k, mZ^{a_1 k})$.
- **Re-Encryption (R).** Anyone can change a second-level ciphertext for A into a first-level ciphertext for B with $rk_{A\to B} = g^{a_1 b_2}$. From $c_{a,r} = (g^k, mZ^{a_1 k})$, compute $e(g^k, g^{a_1 b_2}) = Z^{b_2 a_1 k}$ and publish $c_{b,2} = (Z^{b_2 a_1 k}, mZ^{a_1 k}) = (Z^{b_2 k'}, mZ^{k'})$.
- **Decryption (D_1, D_2).** To decrypt a first-level ciphertext $c_{a,i} = (\alpha, \beta)$ with secret key sk_a, output $\beta/\alpha^{1/a_i} = m$ for $i \in \{1, 2\}$.

Discussion of Scheme. This scheme is similar to the previous one, except to accept delegations, a user must store two secret keys. The security of this scheme relies on an extension of the decisional bilinear Diffie-Hellman (DBDH) assumption [7, 10]; the proof of Boneh and Franklin [7] that the DBDH problem is hard in generic groups, in the sense of Shoup [36], can be easily extended to this problem, when one recalls that the additional parameter $e(g,g)^{bc^2}$ is represented as a random string in the range of the mapping. Furthermore, we note that the semantic security for a user who accepts, but does not give delegations can be reduced to the CoDDH problem; that is, given (g, g^a, Z^b, Q) for random $a, b \leftarrow \mathbb{Z}_q$ and an element $Q \in G_2$, decide if $Q = Z^{ab}$. Original first-level ciphertexts of the form $(Z^{a_2 k}, mZ^k)$ are exactly like El Gamal [16] and thus their security only depends on DDH in G_2. However, under the CoDDH assumption, Alice's security is assured even when people send her second-level ciphertexts (not knowing that she isn't making any delegations). Proof of Theorem 3.1 is in Appendix B.

Theorem 3.1 *The above scheme is correct and secure assuming that for random $g \leftarrow G_1$, $a, b, c \leftarrow \mathbb{Z}_q$, and $Q \in G_2$, given $(g, g^a, g^b, g^c, e(g,g)^{bc^2}, Q)$ it is hard to decide if $Q = e(g,g)^{abc}$ (standard security) and the discrete logarithm assumption (master secret security).*

3.2. Temporary Unidirectional Proxy Re-encryption

In addition to the global parameters (g, Z), suppose there is a trusted server that broadcasts a random value $h_i \in G_1$ for each time period $i \geq 1$ for all users to see. Let $Z_i = e(g, h_i) \in G_2$. We enable Alice to delegate to Bob only for time period i, say, while she is on vacation, as follows.

- **Key Generation (KG).** A user A's key pair is of the form $pk_a = (g^{a_0}, g^{a_r})$, $sk_a = (a_0, a_r)$, (plus a *temporary* secret a_i for time period i which will be generated in RG).
- **Re-Encryption Key Generation (RG).** A user A *publicly* delegates to B *during time period* i as follows: (1) B chooses and stores a random value $b_i \in \mathbb{Z}_q$, and publishes $h_i^{b_i}$; then, (2) A computes and publishes $rk^i_{A\to B} = h_i^{a_r b_i / a_0}$.
- **First-Level Encryption (E_1).** To encrypt $m \in G_2$ under pk_a during time period i, output $c_{a,0} = (Z_i^{a_0 k}, mZ_i^k)$.
- **Second-Level Encryption (E_2).** To encrypt $m \in G_2$ under pk_a during time period i, output $c_{a,i} = (g^{a_0 k}, mZ_i^{a_r k})$.
- **Re-Encryption (R).** Anyone can change a second-level ciphertext for A into a first-level ciphertext for B with $rk_{A\to B,i} = h_i^{a_r b_i / a_0}$. From $c_{a,i} =$

$(g^{a_0 k}, mZ_i^{a_r k})$, compute $e(g^{a_0 k}, rk_{A \to B}) = Z_i^{b_i a_r k}$ and publish $c_{b,i} = (Z_i^{b_i a_r k}, mZ_i^{a_r k}) = (Z_i^{b_i k'}, mZ_i^{k'})$.

- **Decryption (D_1).** To decrypt $c_{a,i} = (\alpha, \beta)$, compute $m = \beta / \alpha^{1/a_i}$ for $a_i \in \{a_0, a_1, a_2, \ldots\}$.

Discussion of Scheme. A single *global* change can invalidate all previous delegations without *any* user needing to change their public key. Proof of Theorem 3.2 appears in the full version of this paper.

Theorem 3.2 *The above scheme is correct and secure assuming that for random $g \leftarrow G_1$, $a, b, c \leftarrow \mathbb{Z}_q$ and $Q \in G_2$, given (g, g^a, g^b, g^c, Q) it is hard to decide if $Q = e(g,g)^{ab^2/c}$ (standard security) and the discrete logarithm assumption (master secret security).*

4. Encrypted File Storage

Our file system uses an untrusted *access control server* to manage access to encrypted files stored on distributed, untrusted block stores. We use proxy re-encryption to allow for access control without granting full decryption rights to the access control server.

To our knowledge, no experimental implementation of proxy re-encryption exists, which makes it difficult to argue about the effectiveness of the proxy re-encryption primitive.

Overview. In our file system, end users on client machines wish to obtain access to integrity-protected, confidential content. A content owner publishes encrypted content in the form of a many-reader, single-writer file system. The owner encrypts blocks of content with unique, symmetric *content keys*. A content key is then encrypted with an asymmetric master key to form a *lockbox*. The lockbox resides with the block it protects.

Untrusted block stores make the encrypted content available to everyone. Users download the encrypted content from a block store, then communicate with an access control server to decrypt the lockboxes protecting the content. The content owner selects which users should have access to the content and gives the appropriate delegation rights to the access control server.

Access Control Using Proxy Cryptography. We propose an improvement on the access control server model that reduces the server's trust requirements by using proxy cryptography. In our approach, the keys used to encrypt files are themselves securely encrypted under a master public key, using one of the schemes in Section 3. Because the access control server does not possess the master secret, it cannot be corrupted so as to gain access to the file keys and access encrypted files. The secret master secret key remains offline, in the care of a content owner who uses it only to generate the re-encryption keys used by the access control server. When an authorized user requests access to a file, the access control server uses proxy cryptography to directly re-encrypt the appropriate content key(s) from the master public key to the user's public key.

This architecture has significant advantages over systems with trusted access control servers. The key material stored on the access control server cannot be used to access stored files, which reduces the need to absolutely trust the server operator, and diminishes the server's value to attackers. The master secret key itself is only required by a content owner when new users are added to the system, and can therefore be stored safely offline where it is less vulnerable to compromise. Finally the schemes in Section 3 are *unidirectional*, meaning that users do not need to reveal or otherwise compromise their secret keys in order to join the system. This allows content owners to add users to the system without interaction, simply by obtaining their public key. Because this system works with users' long-term keys (rather than generating ephemeral keys for the user), there is an additional incentive for users not to reveal their decryption keys.

The proposed design fundamentally changes the security of an access control server storage system. In this new model, much of the security relies on the strength of a provably-secure cryptosystem, rather than on the trust of a server operator for mediating access control. Because the access control server cannot successfully re-encrypt a file key to a user without possessing a valid delegation key, the access control server cannot be made to divulge file keys to a user who has not been specifically authorized by the content owner, unless this attacker has previously stolen a legitimate user's secret key.

Chefs. We implemented our file system on top of Chefs [19], a confidentiality-enabled version of the SFS Read-Only file system [20]. Chefs is a single-writer, many-reader file system that provides decentralized access control in integrity-protected content distribution. A content owner creates a signed, encrypted database from a directory tree of content. The database is then replicated on untrusted hosts (e.g., volunteers). A client locates a replica, then requests the encrypted blocks.

Chefs tags each content block with a lockbox. The lockbox contains a 128-bit AES key, itself encrypted

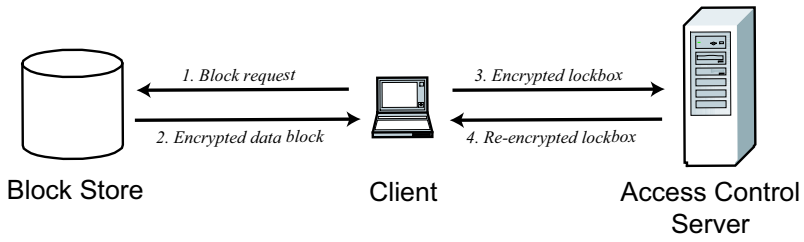

Figure 1. Typical operation of the proxy re-encryption file system. The user's client machine fetches encrypted blocks from the block store. Each block includes a lockbox encrypted under a master public key. The client then transmits lockboxes to the access control server for re-encryption under the user's public key. If the access control server possesses the necessary re-encryption key, it re-encrypts the lockbox and returns the new ciphertext. The client can then decrypt the re-encrypted block with the user's secret key.

with a shared group AES key. Chefs assumes an out-of-band mechanism for content owners to distribute group keys to users.

We chose the Chefs architecture because it allowed us to experiment with different granularities of encryption (per-file and per-directory) while providing a transparent file system interface for our experiments.

4.1. Design and Implementation

We modified Chefs to include an access control server. Every block in a Chefs database is encrypted with a 128-bit AES content key in CBC mode. Depending on the granularity of the encryption, a content key can be shared across all of the blocks in a particular file, directory or database, or unique keys can be used for each block. Content keys are themselves encrypted under a system master key using the third bilinear El Gamal scheme from Section 3.1. This encryption results in a set of lockboxes stored with the file data, either in file or directory inodes (per-file and per-directory encryption) or within the blocks themselves (per-block encryption). The proxy re-encryption makes the sealed lockbox 512 bits, even though it still only protects a 128-bit AES key.

When a client encounters a block for which it does not possess a content key, it asks the access control server to re-encrypt the lockbox from the master key to the client's public key. If the access control server possesses an appropriate re-encryption key for this client and master key, it performs the appropriate proxy re-encryption and returns the result to the client, which can then decrypt the lockbox. Figure 1 illustrates this procedure.

Each re-encryption call necessarily results in a round-trip network request, in addition to the proxy re-encryption and client-side decryption of the re-encrypted ciphertext. Thus, the choice of encryption granularity greatly affects the number of re-encryption calls made from the client to the access control server, which in turn affects the performance of the system.

4.2. Experimental Results

In implementing a proxy re-encryption file system, we had two goals in mind. First, we wished to show that proxy re-encryption could be successfully incorporated into a basic cryptographic file system. Second, we sought to prove that the additional security semantics provided by a proxy re-encrypting access control server came at an acceptable cost to system performance.

To achieve this second goal, we conducted a number of benchmarks using the proxy-enabled Chefs file system, using various granularities of content key usage (per-block and per-file). Along with these experiments, we conducted microbenchmarks of the proxy re-encryption functions used in our implementation, as well as application-level benchmarks measuring file system performance. To provide a standard of comparison, we conducted the same experiments on an unmodified Chefs configuration with no access control server or proxy re-encryption, using only a single preset AES key to secure the contents of the database.

Experimental Setup. For the purposes of our testing, we used two machines to benchmark the proxy-enabled Chefs file system. The client machine consisted of an AMD Athlon 2100+ 1.8 GHz with 1 GB RAM and an IBM 7200 RPM, 40 GB, Ultra ATA/100 hard drive. The server machine was an Intel Pentium 4 2.8 GHz with 1 GB RAM and a Seagate Barracuda 7200 RPM, 160 GB, Ultra ATA/100 hard drive. Both

Parameter size	*Encryption*	*Decryption (by original recipient)*	*Re-encryption*	*Decryption (by delegatee)*
256-bit	3.1 ms	8.6 ms	8.4 ms	1.5 ms
512-bit	7.7 ms	21.9 ms	21.7 ms	3.4 ms

Table 2. Average operation times for the bilinear El Gamal proxy re-encryption scheme on our client machine. All operations refer to re-encryptable "second-level" ciphertexts.

systems were running Debian testing/unstable. The client and the server were situated in different cities, representing a distributed file system scenario. We measured the round-trip latency between the two machines at 13 ms, and the maximum sustained throughput of the network link at 7 Mbit/sec. We implemented the cryptographic primitives for the bilinear El Gamal scheme using version 4.83 of the MIRACL cryptographic library [35], which contains efficient implementations of the Tate pairing as well as fast modular exponentiation and point multiplication.

Cryptographic Benchmark. Average times for the cryptographic operations in the bilinear proxy re-encryption scheme (the third one from Section 3.1) are given in Table 2, and provide some basis for understanding the impact of the proxy re-encryption on overall file system performance. These results indicate that re-encryption is the one of the most time-consuming of the cryptographic operations performed in our file system. In our file system, we use 512-bit proxy re-encryption to protect content keys in lockboxes.

We conducted our benchmarks using the 512-bit parameter size for the proxy re-encryption scheme, and various encryption granularities, including per-block and per-file. Our microbenchmarks, presented in Figures 2 and 3, include runs of the small-file and large-file tests from the LFS suite of file system performance tests [33]. We use the read phases of the LFS test to measure the fundamental performance of our system.

The first test consists of a sequential read of a large file. The second test reads several small files. These two tests capture common workloads in a typical file system. For each of these tests, we experimented with different encryption granularities, including per-block and per-file settings. The small file benchmark in particular is a worst-case scenario for a proxy-enabled file system, as it requires a large number of lockbox re-encryptions relative to the amount of data read. On the other hand, the large-file case tends to exhibit exactly the opposite effect, as the ratio of re-encryptions to data read is much smaller. In general, all per-block encryption scenarios tend to be the least efficient (and least practical) when proxy re-encryption is enabled.

Small-file Benchmark. The SFSRO and Chefs benchmarks each generate 2,022 RPCs to fetch content from the block store (1,000 files, 10 directories, and one root directory — each generating two RPCs: one for the inode, one for the content).

Note that Chefs adds virtually no discernible overhead, even though the client decrypts every content fetch with 128-bit AES in CBC mode. With the round-trip time accounting for at least 26 seconds of the measurement, the network overshadows the cost of cryptography.

The proxy re-encryption file system first makes 2,022 fetches of content, just like Chefs. With per-file granularity of content keys, the small-file benchmark generates 1,011 re-encryption RPCs. The proxy re-encryption file system takes about 44 seconds longer than Chefs. We attribute 39 seconds of this difference to the 13 ms round-trip time, 21.7 ms re-encryption time on the server, and 3.4 ms delegatee decryption time on the client for each RPC (See Table 2). We hope to explain the remaining 5 seconds by conducting measurements on isolated portions of our client code.

With per-block granularity, the small-file benchmark generates 2,022 re-encryption RPCs. A file or directory consists of an inode and data block, thus each read now generates two re-encryptions. The proxy re-encryption file system takes about 87 seconds longer than Chefs. Because the per-block re-encryption generates twice as many re-encryption RPCs as the per-file scenario, the results concur with our expectations.

Large-file Benchmark. The large-file benchmark generates 5,124 RPCs to fetch 40 MB of content from the block store (two RPCs for the root directory, two for the file, and 5,120 for the file data). In the SFSRO and Chefs experiments, the 7 MBit bandwidth largely dominates the throughput.

Because the large-file workload involves only a single file, the per-file proxy re-encryption has no discernible

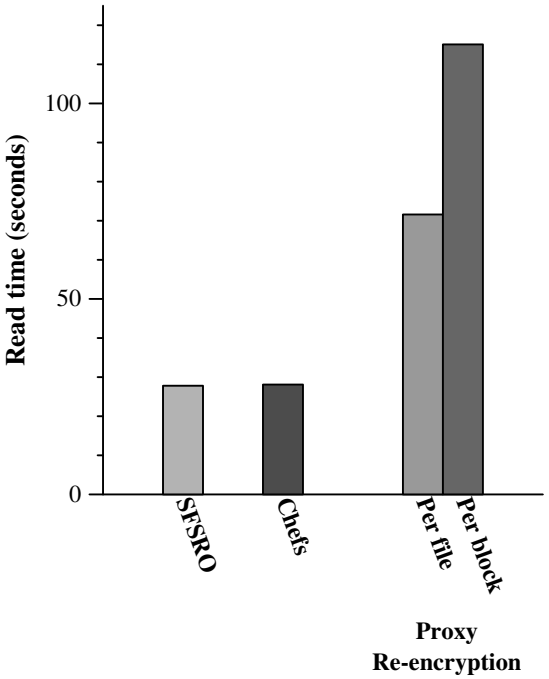

Figure 2. Small-file microbenchmark from LFS suite. We perform a complete read on 1,000 1 KB files dispersed in 10 directories.

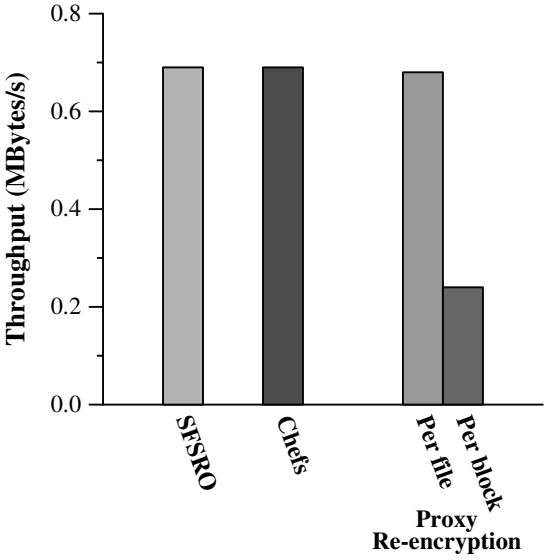

Figure 3. Large-file microbenchmark from LFS suite. We perform a sequential read on a 40 MB file in 8 KB blocks.

cost. There are a mere two proxy re-encryption RPCs (one for the root, one for the file). The per-block proxy re-encryption generates 5,124 re-encryption RPCs, thus we expect a significant degradation of throughput because of the number of extra round-trips.

The cost of per-block re-encryption is prohibitively expensive for large files. We expect that per-file granularity or per-file-system granularity will be much more common than per-block granularity. For instance, we do not expect users to grant access to portions of a single file. Rather, we expect users would share access-controlled content in collections of files — similar to a collection of Web pages or a directory subtree.

Application-level Benchmark. Our application-level benchmark consists of an Emacs version 21.3 compilation. The source code is stored in our file system, while the resulting binaries are written to a local disk. This workload requires access to approximately 300 files. The results of this test are presented in Figure 4, and show that the per-file and even per-block proxy cryptography adds negligible overhead for this application workload. We believe the cost is nominal for the additional security semantics of proxy re-encryption.

Scalability. We also measured how well the access control server performs under a heavy load. Figure 5 shows that our proxy re-encryption server can scale up to 1,000 pending requests before exhibiting signs of stress. We replayed a trace of proxy re-encryption RPCs. This required no computation on the client side, but caused the server to perform proxy re-encryption. We start by issuing a single request, waiting for the response before issuing another request. To simulate many simultaneous clients, we gradually increase the window size of outstanding RPCs.

Our server is able to sustain 100 re-encryptions/sec until reaching about 1,000 outstanding requests. The server coped with up to 10,000 outstanding re-encryption requests, but quickly spiraled downwards thereafter. Note that our server is faster than the client machine, able to perform a single proxy re-encryption in 9 ms.

4.3. Discussion

Our access control server acts like a credentials download service. For instance, PDM [31] stores encrypted credentials on a server. A user decrypts the credentials with a password. PDM works fine when an encrypted credential is available to a single individual. However, our file system supports group access control. We could

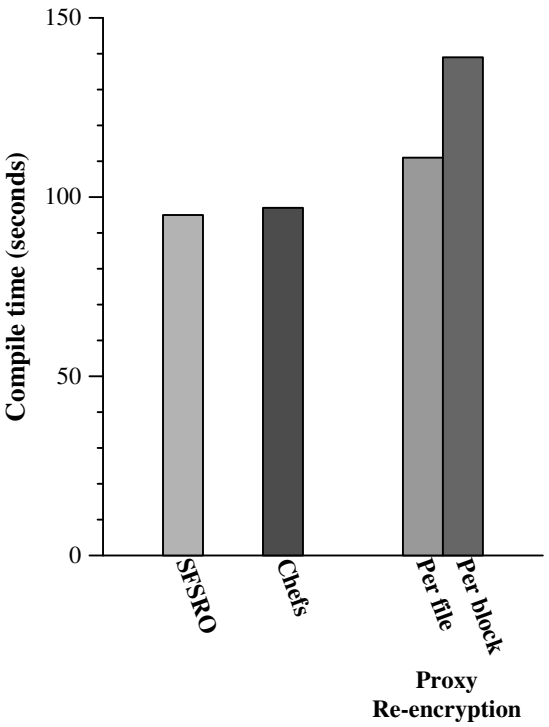

Figure 4. Application-level benchmark. We record the time to compile Emacs version 21.3.

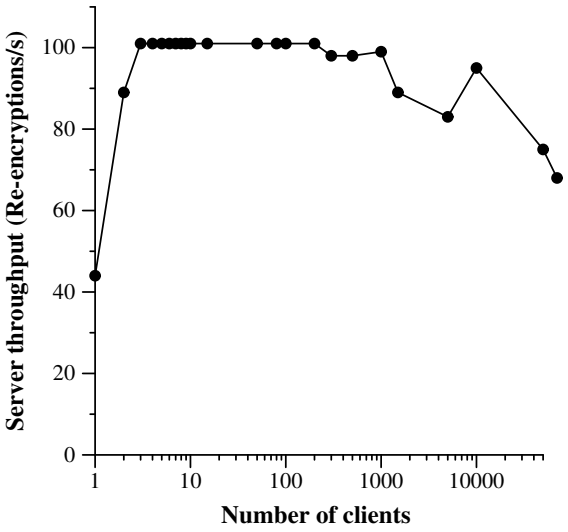

Figure 5. Aggregate access control server throughput. The server can tolerate 1,000 simultaneous re-encryption requests before showing signs of saturation.

use PDM instead of our access control server, but this would reduce the key distribution problem to that of sharing a password with all group members.

We selected a single-writer, many-reader file system rather than a general purpose file system to experiment with proxy re-encryption in content distribution. This eliminates problems not directly related to proxy re-encryption, such as fork consistency [29].

In practice, an organization's data may consist of many distinct file sets or equivalence classes, access to each of which should be limited to a subset of the organization's members. For instance, a large company with several departments might wish to keep data from individual departments confidential within the originating department. However, an access control server shared with other departments would have advantages in reliability and logging. This can easily be achieved by using many different master keys, each of which encrypts content keys for files owned to a different group. The corresponding secret keys can be held by different content owners, whose only operational responsibility is to generate re-encryption keys for new users.

Because there is no fundamental difference in format between a master public key and a user's public key, individual users can use their own public keys as master keys, allowing users to act as content owners of their own personal file sets. Additional possibilities can be achieved if multiple file keys are used to encrypt single files, allowing for files that are available only to users who belong to multiple groups simultaneously.

We believe that our experimental results demonstrate the practicality of proxy re-encryption in protecting stored content. Though proxy re-encryption adds a level of overhead to file system, this overhead is not extreme, and can be worth the additional security that comes from centralizing access control at a semi-trusted access control server. Various system choices, such as parameter sizes and encryption granularity can greatly affect the efficiency of the system; we have selected the ones we believe to be most promising.

5. Conclusions

In this paper, we explored proxy re-encryption from a predominately systems perspective. We outlined the characteristics and security guarantees of previously known schemes, and compared them to a suite of improved re-encryption schemes we present over bilinear maps. These pairing-based schemes realize important new features, such as safeguarding the master secret key of the delegator from a colluding proxy and delegatee. One of the most promising applications for proxy re-

encryption is giving proxy capabilities to the key server of a confidential distributed file system; this way the key server need not be fully trusted with all the keys of the system and the secret storage for each user can also be reduced. We implemented this idea in the context of the SFSRO file system, and showed experimentally that the additional security benefits of proxy re-encryption can be purchased for a manageable amount of run-time overhead. We leave open the theoretical problem of finding a proxy re-encryption scheme that does not allow further delegations; that is, Bob (plus the proxy) can not delegate to Carol what Alice has delegated to him. We also leave open the practical problems of finding more efficient implementations of secure proxy re-encryption schemes, as well as conducting more experimental tests in other applications.

Chefs is part of the SFSRO code base available via CVS from `www.fs.net`. Source code for our proxy re-encryption library and file system is available upon email request.

Acknowledgments. We are grateful to Srinath Anantharaju, Jan Camenisch, Frans Kaashoek, Mahesh Kallahalla, Ron Rivest, and the anonymous reviewers for helpful comments and discussions. Thanks also to David Liben-Nowell for his LaTeX expertise. This work was partially supported by an NDSEG Graduate Research Fellowship, an Intel PhD Fellowship, and a NSF grant.

References

[1] 104th United States Congress. Health Insurance Portability and Accountability A (HIPPA), 1996. http://aspe.hhs.gov/admnsimp/pl104191.htm; Last access: August 16, 2004.

[2] A. Adya, W. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. *SIGOPS Oper. Syst. Rev.*, 36(SI):1–14, 2002.

[3] J. H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In *Proceedings of Eurocrypt '02*, volume 2332 of LNCS, pages 83–107, 2002.

[4] J. Baek, R. Steinfeld, and Y. Zheng. Formal proofs for the security of signcryption. In *Proceedings of Public Key Cryptography '02*, volume 2274 of LNCS, pages 80–98, 2002.

[5] M. Blaze. A cryptographic file system for UNIX. In *ACM Conference on Computer and Communications Security*, pages 9–16, 1993.

[6] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In *Proceedings of Eurocrypt '98*, volume 1403, pages 127–144, 1998.

[7] D. Boneh and M. Franklin. Identity-based encryption from the Weil Pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.

[8] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures. In *Proceedings of Eurocrypt '03*, volume 2656 of LNCS, pages 416–432, 2003.

[9] D. Boneh, H. Shacham, and B. Lynn. Short signatures from the Weil pairing. In *Proceedings of Asiacrypt '01*, volume 2248, pages 514–532, 2001.

[10] J. H. Cheon and D. H. Lee. Diffie-Hellman problems and bilinear maps. *Cryptology ePrint Archive: Report 2002/117*, 2001.

[11] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *Proceedings of Eurocrypt '02*, volume 2332 of LNCS, pages 45–64, 2002.

[12] Y. Dodis, M. K. Franklin, J. Katz, A. Miyaji, and M. Yung. Intrusion-resilient public-key encryption. In *Proceedings of CT-RSA '03*, volume 2612 of LNCS, pages 19–32, 2003.

[13] Y. Dodis, M. K. Franklin, J. Katz, A. Miyaji, and M. Yung. A generic construction for intrusion-resilient public-key encryption. In *Proceedings of CT-RSA '04*, volume 2964 of LNCS, pages 81–98, 2004.

[14] Y. Dodis and A. Ivan. Proxy cryptography revisited. In *Proceedings of the Tenth Network and Distributed System Security Symposium*, February 2003.

[15] Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-insulated public key cryptosystems. In *Proceedings of Eurocrypt '02*, volume 2332 of LNCS, pages 65–82, 2002.

[16] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of Crypto '84*, pages 10–18, 1984.

[17] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *proceedings of Crypto '86*, volume 263 of LNCS, pages 186–194, 1986.

[18] K. Fu. Group sharing and random access in cryptographic storage file systems. Master's thesis, Massachusetts Institute of Technology, May 1999.

[19] K. Fu. *Integrity and access control in untrusted content distribution networks*. PhD thesis, Massachusetts Institute of Technology, Manuscript.

[20] K. Fu, M. F. Kaashoek, and D. Mazières. Fast and secure distributed read-only file system. *ACM Trans. Comput. Systems*, 20(1):1–24, 2002.

[21] S. D. Galbraith, K. Harrison, and D. Soldera. Implementing the tate pairing. In *Proceedings of the Algorithmic Number Theory Symposium*, volume 2369 of LNCS, pages 324–337, 2002.

[22] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh. SiRiUS: Securing remote untrusted storage. In *Proceedings of the Tenth Network and Distributed System Security Symposium*, pages 131–145, February 2003.

[23] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[24] P. Golle, M. Jakobsson, A. Juels, and P. F. Syverson. Universal Re-encryption for Mixnets. In *Proceedings of CT-RSA '04*, volume 2964 of LNCS, pages 163–178, 2004.

[25] A. Harrington and C. Jensen. Cryptographic access control in a distributed file system. In *Proceedings of 8th ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, Villa Gallia, Como, Italy, June 2003. ACM.

[26] M. Jakobsson. On quorum controlled asymmetric proxy re-encryption. In *Proceedings of Public Key Cryptography*, pages 112–121, 1999.

[27] A. Joux. A one-round protocol for tripartite Diffie-Hellman. In *Proceedings of ANTS-IV conference, Lecture Notes in Computer Science.*, volume 1838, pages 385–394, 2000.

[28] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus – scalable secure file sharing on untrusted storage. In *Proceedings of the Second USENIX Conference on File and Storage Technologies*, March 2003.

[29] J. Li, M. N. Krohn, D. Mazières, and D. Shasha. Secure untrusted data repository (SUNDR). In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, pages 91–106, San Francisco, CA, December 2004.

[30] M. Mambo and E. Okamoto. Proxy Cryptosystems: Delegation of the Power to Decrypt Ciphertexts. *IEICE Trans. Fund. Electronics Communications and Computer Science*, E80-A/1:54–63, 1997.

[31] R. Perlman and C. Kaufman. PDM: A new strong password-based protocol. In *Proceedings of the 10th USENIX Security Symposium*, August 2001.

[32] D. Reed and L. Svobodova. Swallow: A distributed data storage system for a local network. In A. West and P. Janson, editors, *Local Networks for Computer Communications*, pages 355–373. North-Holland Publishing Company, Amsterdam, 1981.

[33] M. Rosenblum and J. Ousterhout. The design and implementation of a log-structured file system. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP)*, pages 1–15, Pacific Grove, CA, October 1991. ACM.

[34] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptography*, 4:161–174, 1991.

[35] M. Scott. MIRACL library. Indigo Software. http://indigo.ie/∼mscott/#download.

[36] V. Shoup. Lower bounds of discrete logarithms and related problems. In *Proceedings of Eurocrypt '97*, volume 1233 of LNCS, pages 256–266, 1997.

[37] F. Zhang, R. Safavi-Naini, and W. Susilo. An efficient signature scheme from bilinear pairings and its applications. In *Proceedings of Public Key Cryptography '04*, volume 2947 of LNCS, pages 277–290, 2004. Full version of the paper available at http://www.uow.edu.au/∼fangguo/PKC04.pdf.

[38] Y. Zheng. Signcryption and its applications in efficient public key solutions. In *Proceedings of ISW '97*, volume 1396 of LNCS, pages 291–312, 1997.

[39] L. Zhou, M. A. Marsh, F. B. Schneider, and A. Redz. Distributed blinding for ElGamal re-encryption. Technical Report 2004–1924, Cornell Computer Science Department, 2004.

A. Definition of Unidirectional Proxy Encryption

Definition A.1 (Unidirectional Proxy Re-encryption) *A* unidirectional proxy re-encryption scheme *is a tuple of (possibly probabilistic) polynomial time algorithms* $(KG, RG, \vec{E}, R, \vec{D})$*, where the components are defined as follows:*

- $(KG, \vec{E}, \vec{D})$ are the standard key generation, encryption, and decryption algorithms for the underlying cryptosystem. Here $\vec{E}$ and $\vec{D}$ are (possibly singleton) sets of algorithms. On input the security parameter 1^k, KG outputs a key pair (pk, sk). On input pk_A and message $m \in M$, for all $E_i \in \vec{E}$ the output is a ciphertext C_A. On input sk_A and ciphertext C_A, there exists a $D_i \in \vec{D}$ that outputs the message $m \in M$.
- On input $(pk_A, sk_A^{\dagger}, pk_B, sk_B^{*})$, the re-encryption key generation algorithm, RG, outputs a key $rk_{A\to B}$ for the proxy. The fourth input marked with a '$*$' is sometimes omitted; when this happens we say that RG is *non-interactive* since the delegatee does not need to be involved in the generation of the re-encryption keys. The second input marked with a '†' may be replaced by the tuple $(rk_{A\to C}, sk_C)$; see Remark A.3 for more.
- On input $rk_{A\to B}$ and ciphertext C_A, re-encryption function, R, outputs C_B.

Correctness. Informally, a party holding a secret key sk_A should always be able to decrypt ciphertexts encrypted under pk_A; while a party B should be able to decrypt $R(rk_{A\to B}, C_A)$. $\vec{E}$ may contain multiple encryption algorithms; for example, having *first-level* encryptions that cannot be re-encrypted by the proxy; while *second-level* encryptions can be re-encrypted by the proxy and then decrypted by delegatees. This provides the sender with a choice *given the same public key* whether to encrypt a message only to Alice or to Alice and, say, her secretary. Whenever a re-encryption does take place, however, we require that the underlying plaintext remain *consistent* – i.e., Bob should get exactly what Alice was supposed to receive.[2]

[2]Note, this only applies to ciphertexts that were honestly generated by the sender; no guarantee is implied in the case of malformed ciphertexts.

More formally, let key pairs (pk_A, sk_A) and (pk_B, sk_B), generated according to KG, belong to parties A and B, respectively, and let $rk_{A\to B}$ be generated according to RG. Then, for all messages m in the message space M, the following equations hold with probability one:

$$\begin{aligned}&\forall E_i \in \vec{E},\ \exists D_j \in \vec{D},\\ &\quad D_j(sk_A, E_i(pk_A, m)) = m,\\ &\exists E_i \in \vec{E},\ \exists D_j \in \vec{D},\\ &\quad D_j(sk_B, R(rk_{A\to B}, E_i(pk_A, m))) = m.\end{aligned}$$

We provide a security definition similar to that of Ivan and Dodis [14]. Although their definition was for CCA2 security, they instead used CPA security for the El Gamal, RSA, and IBE-based schemes; for simplicity, we focus directly on CPA security. The first main difference between our definitions is that we consider the security of a user against a *group* of colluding parties; for example, the security of a delegator against the proxy and many delegatees, whereas the Ivan-Dodis definition focused on a single delegatee. Secondly, we discuss the system's security for circular delegation where the adversary watches Alice and Bob delegate to each other. Finally, we provide a new guarantee for the delegator – even if the proxy and all delegatees collude, they can not recover his master secret key. We discuss some benefits of this last feature in Remark A.4.

Definition A.2 (Security of U. P. Re-encryption) Let $\Gamma = (KG, RG, \vec{E}, R, \vec{D})$ be a unidirectional proxy re-encryption scheme.

Standard Security. The underlying cryptosystem $(KG, \vec{E}, \vec{D})$ is semantically-secure [23] against *anyone* who has not been delegated the right to decrypt. That is, for all PPT algorithms A_k, $E_i \in \vec{E}$, and $m_0, m_1 \in M_k$,

$$\begin{aligned}\Pr[(pk_B, sk_B) \leftarrow KG(1^k), \{(pk_q, sk_q) \leftarrow KG(1^k)\},\\ \{rk_{q\to B} \leftarrow RG(pk_q, sk_q, pk_B, sk_B^*)\},\\ \{(pk_h, sk_h) \leftarrow KG(1^k)\},\\ \{rk_{B\to h} \leftarrow RG(pk_B, sk_B, pk_h, sk_h^*)\},\\ \{rk_{h\to B} \leftarrow RG(pk_h, sk_h, pk_B, sk_B^*)\},\\ (m_0, m_1, \alpha) \leftarrow A_k(pk_B, \{(pk_q, sk_q)\}, \dots\\ \dots \{pk_h\}, \{rk_{q\to B}\}, \{rk_{B\to h}\}),\\ b \leftarrow \{0,1\}, b' \leftarrow A_k(\alpha, E_i(pk_B, m_b)) :\\ b = b'] = \nu(k).\end{aligned}$$

Master Secret Security. The long term secrets of a delegator (sometimes serving as a delegatee) cannot be computed or inferred by even a coalition of colluding delegatees. For all PPT algorithms A_k,

$$\begin{aligned}\Pr[(pk_B, sk_B) \leftarrow KG(1^k), \{(pk_q, sk_q) \leftarrow KG(1^k)\},\\ \{rk_{B\to q} \leftarrow RG(pk_B, sk_B, pk_q, sk_q^*)\},\\ \{rk_{q\to B} \leftarrow RG(pk_q, sk_q, pk_B, sk_B^*)\},\\ \alpha \leftarrow A_k(pk_B, \{(pk_q, sk_q)\}, \{rk_{B\to q}\}, \{rk_{q\to B}\}) :\\ \alpha = sk_B] = \nu(k).\end{aligned}$$

Remark A.3 Unfortunately, acheiving security based on the definition of the re-encryption key generation function RG as originally stated is very difficult to realize. We do not know of any such scheme, including the prior work of Ivan and Dodis [14], that does not succumb to the follow attack: *transfer of delegation rights*, where, on input sk_B and $rk_{A\to B}$, one can compute $rk_{A\to C}$. (Recall our discussion of non-transferability in Section 3.) To see this in our second and third schemes, consider that on input b and $g^{b/a}$, one can output $(g^{b/a})^{1/b} = g^{1/a}$ which would allow anyone to decrypt Alice's second-level ciphertexts. Thus, we modify the definition of RG to be executed with *either* the secret key of the delgator Alice sk_A or with both a re-encryption key from Alice to Bob $rk_{A\to B}$ and Bob's secret key sk_B. This implies that Bob is *allowed* to transfer Alice's decryption capability. Arguably, this relaxed definition is not so damaging since Alice is already trusting Bob enough to delegate decryption rights to him.

Remark A.4 At first glance, master secret security may seem very weak. All it guarantees is that an adversary cannot output a delegator's secret key sk_A. One might ask why this is useful. Recall, however, that most standard signature schemes, such as El Gamal [16] and Schnorr [34], are actually proofs of knowledge of a discrete logarithm value, such as $sk_A = a \in \mathbb{Z}_q$, turned into a signature using the Fiat-Shamir heuristic [17]. Intuitively, if an adversary cannot output Alice's secret key, then the adversary cannot prove knowledge of it either. Thus, using a proxy re-encryption scheme with master secret security means that a user may be able to safely delegate decryption rights (via releasing g^a) without delegating signing rights for the *same public key* Z^a.

B. Proofs of Security

Theorem 3.1 *The scheme in Section 3.1 is correct and secure under the extended decisional bilinear Diffie-Hellman* (EDBDH) *assumption that is for random* $g \leftarrow G_1$, $a, b, c \leftarrow \mathbb{Z}_q$, *and* $d \in \mathbb{Z}_q$, *given* $(g, g^a, g^b, g^c, Z^{bc^2}, Z^d)$ *it is hard to decide if* $d \equiv abc$

mod q (standard security) and the discrete logarithm assumption (master secret security).

Proof. Our security definition quantifies over all encryption algorithms $E_i \in \vec{E}$; in this case, we have two algorithms E_1, E_2, where a ciphertext of the first form $(Z^k, mZ^{a_1 k})$ can be publicly computed from a ciphertext of the second form $(g^k, mZ^{a_1 k})$ via $e(g, g^k) = Z^k$. Thus, it suffices to argue the security of E_2 only.

Standard Security.[3] Suppose A distinguishes encryptions of E_2 with non-negligible probability, we simulate an adversary S that decides eDBDH as follows:

1. On input $(g, g^a, g^b, g^c, Z^{bc^2}, Z^d)$, the simulator sets $y = g^c$, $W = e(y, y)$, and obtains the tuple:

$$(y = g^c, y^\alpha = g^a, y^\beta = g^b, y^\gamma = g,$$
$$W^{\beta/\gamma} = Z^{bc^2}, W^{\alpha\beta/\gamma} = Z^d),$$
$$\text{for } \alpha = \tfrac{a}{c}, \beta = \tfrac{b}{c}, \gamma = \tfrac{1}{c}.$$

   The simulator sends A the global parameters (y, W) and the target public key $pk_T = (W^{\beta/\gamma}, g^t)$, where t is randomly selected from $\mathbb{Z}_q$.
2. Next, for $i = 1$ up to $\text{poly}(k)$, A can request:
   (a) a delegation from T to an honest party. S randomly selects $r_{(i,1)}, r_{(i,2)} \leftarrow \mathbb{Z}_q$, sets $rk_{T\to i} \leftarrow y^{\gamma r_{(i,2)}}$ and $pk_i = (W^{r_{(i,1)}}, y^{\beta r_{(i,2)}})$, and sends $(pk_i, rk_{T\to i})$ to A.
   (b) a delegation to T from an honest party. S uses either the recorded value $r_{(i,1)}$ from the previous step if the honest party already exists, or generates fresh random values for a new party, and computes $rk_{i\to T} = (g^t)^{r_{(i,1)}}$.
   (c) a delegation to T from a party corrupted by A. A can generate these delegations internally by running $(pk_i, sk_i) \leftarrow KG(1^k)$ and computing $rk_{i\to T} = (g^t)^{i_1}$.
3. Eventually, A must output a challenge (m_0, m_1, τ), where $m_0 \neq m_1 \in M$ and τ is its internal state information. The simulator randomly selects $b \in \{0, 1\}$, computes the ciphertext $c_b = (y^\alpha, m_b W^{\alpha\beta/\gamma})$, sends (c_b, τ) to A, and waits for A to output $b' \in \{0, 1\}$.
4. If $b = b'$, then S guesses "$d = abc$"; otherwise S guesses "$d \neq abc$".

[3]The semantic security for a user who accepts, but does not give delegations can be reduced to the CoDDH problem; that is, given (g, g^a, Z^b, Q) for random $a, b \leftarrow \mathbb{Z}_q$ and an element $Q \in G_2$, decide if $Q = Z^{ab}$. Here, we prove security for a more general set of users as in definition A.2.

First, we observe that if $d = abc$, then the simulation is perfect; and if $d \neq abc$, then m_b is information-theoretically hidden from A, since $W^{\alpha\beta/\gamma}$(=Z^d) was chosen independently of y^α=(g^a). Thus, if A succeeds with probability $1/2+\varepsilon$, then S succeeds with probability $1/2+\varepsilon/2$. This contradicts the eDBDH assumption.

Master Secret Security. Suppose an adversary A can recover the secret key of a targeted user T (i.e., $sk_t = (t_1, t_2)$) with non-negligible probability by interacting with T according to the second part of definition A.2, then we can build an adversary S that takes discrete logs in G_1. Let us focus our attention on recovering only the value b_1. Our simulator S works as follows:

1. On input (g, g^a) in G_1, output the global parameters (g, Z) and the target public key $pk_t = (Z^a, g^{t_2})$, where $Z^a = e(g, g^a)$ and t_2 is a random element in $\mathbb{Z}_q$.
2. Next, for $i = 1$ up to $\text{poly}(k)$, A can request:
   (a) a delegation from T to a party corrupted by A. S randomly selects $r_{(i,1)}, r_{(i,2)} \leftarrow \mathbb{Z}_q$, sets $rk_{T\to i} \leftarrow g^{a r_{(i,2)}}$, $pk_i = (W^{r_{(i,1)}}, g^{r_{(i,2)}})$, and $sk_i = (r_{(i,1)}, r_{(i,2)})$, and sends $(pk_i, sk_i, rk_{T\to i})$ to A.
   (b) a delegation to T from a party corrupted by A. A can generate these delegations internally by running $(pk_i, sk_i) \leftarrow KG(1^k)$ and computing $rk_{i\to T} = (g^{t_2})^{i_1}$.
3. Eventually, A must output a purported secret key for T of the form (α, β). The simulator returns the value α.

The simulation is perfect; thus A must not be able to recover the master secret key of T, despite accepting and providing numerous delegations to T, because otherwise, S can efficiently solve the discrete logarithm problem in G_1. □