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 (BBS) 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 Dodis and Ivan, 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 a secure 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. Alice could simply 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.

*An earlier version of this paper appeared in the Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS), February 2005, and a journal version has been accepted for publication in ACM Transactions on Information and System Security (TISSEC).

†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.

We provide the first empirical performance measurements of applications using proxy re-encryption. To demonstrate the practical utility of our proxy re-encryption schemes, we measure an implementation of proxy re-encryption used in a secure file system. Our system uses a centralized *access control server* to manage access to encrypted content 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.

1.1 Proxy Re-encryption Background

A methodology for delegating decryption rights was first introduced by Mambo and Okamoto [34] purely as an efficiency improvement over traditional decrypt-and-then-encrypt approaches.

In 1998, Blaze, Bleumer, and Strauss [7] 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 seeing the underlying plaintext. In their Elgamal 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 leaves several open 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 [30] 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 [42].

Recently, Dodis and Ivan [16] realized *unidirectional* proxy encryption for Elgamal, 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 Elgamal 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 Dodis-Ivan IBE scheme [16] 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 Dodis-Ivan 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 [14, 15, 17] and signcryption [3, 5, 41] 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 [28], which *re-randomizes* ciphertexts instead of changing the *public key* that they are encrypted under.

1.2 Applications of Proxy Re-encryption

Proxy re-encryption has many exciting applications in addition to the previous proposals [7, 16, 30, 42] 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 file systems build confidential storage out of untrusted components by using cryptographic storage [2, 6, 26, 32]. 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 and remembers 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 [36] or CNFS [29] assume an out-of-band mechanism for distributing keys for access control. Other systems such as Cepheus [21] 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 sections describing the theory and implementation of proxy re-encryption. Section 2 gives 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.

2 Definitions

Our protocols are based on bilinear maps [8, 9, 10, 31], which we implemented using the fast Tate pairings [25].

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, $\hat{G}_1$ 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; and (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). (In our scheme descriptions we treat G_1 and $\hat{G}_1$ as the same group. However we recognize that, for some instantiations of the mappings, it is more efficient to let G_1 and $\hat{G}_1$ be distinct groups of size q. Our constructions will work in this setting as well.)

Now, we define what a unidirectional proxy re-encryption scheme is and what minimum security properties it should have. We compare our definition to a similar definition due to Dodis and Ivan [16]. In remarks 2.4 and 2.5, we discuss some of the short-comings and benefits of this definition.

Definition 2.2 (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 $rk_{A\rightarrow B}$ 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 '$\dagger$' may in some cases be replaced by the tuple $(rk_{A\rightarrow C}, sk_C)$; see Remark 2.4 for more.
- On input $rk_{A\rightarrow B}$ and ciphertext C_A, the 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 $R(rk_{A\rightarrow B}, C_A)$. $\vec{E}$ 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.[1]

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:

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

We provide a security definition similar to that of Dodis and Ivan [16]. Although their definition was for CCA2 security, they instead used CPA security for the Elgamal, 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 Dodis-Ivan 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 2.5.

Definition 2.3 (Security of Unidirectional Proxy 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 [27] against *anyone* who has not been delegated the right to decrypt. We use subscript B to denote the target user, x to denote the adversarial users, and h to denote the honest users (other than B). That is, for all PPT algorithms A_k, $E_i \in \vec{E}$, and $m_0, m_1 \in M_k$,

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

The above definition captures B's security, even when the proxy (with knowledge of all the re-encryption keys) and a group of adversarial users (with knowledge of their own secret keys) collude against B – provided that B never delegated decryption rights to any adversarial user.

We now turn our attention to what security can be guaranteed in the case that B *does* delegate decryption rights to an adversarial user. Obviously, in this case, the adversary can simply decrypt and trivially win the game above. However, we are now interested in determining whether or not an adversary (consisting of the

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

proxy and a group of colluding users) can recover B's master secret key. (We will later see some examples where, for the same public key, two types of ciphertexts may be generated: one that can be opened by a delegatee, and one that can only be opened with the master secret key.) Thus, we define here (and later prove) that just because B delegates decryption rights to another party does not mean that B surrenders his digital identity (i.e., the meaningful parts of his secret key).

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. We use subscript B to denote the target user and x to denote the adversarial users. For all PPT algorithms A_k,

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

Remark 2.4 Unfortunately, achieving 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 Dodis and Ivan [16], 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 delegator 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 2.5 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. One motivation, mentioned above, stems from the fact that some proxy re-encryption schemes define two or more types of ciphertext, some of which may only be decrypted using the master secret. A scheme which provides master secret security will protect those ciphertexts even in the event that the proxy and delegatee collude. A second motivation comes from the fact that most standard signature schemes, such as Elgamal [19] and Schnorr [38], 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 [20]. 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.

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 $A \to B$ does not allow re-encryption from $B \to A$.

Property	BBS [7]	DI [16]	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 Dodis-Ivan. $*$ indicates master secret key only. † indicates possible to achieve with additional overhead.

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 [7].)

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, both sender and recipient are aware of the proxy re-encryption protocol but do not know whether the proxy is active, has performed any action or made any changes, or even if it exists (the proxy is indeed "invisible"). More specifically, we allow the sender 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*). At the same time, 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 Dodis-Ivan 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 Elgamal and RSA based schemes [16], 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 Dodis-Ivan, $s = s_1 + s_2$; 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.

The property of collusion "safeness" is extremely useful in our context since we allow the sender to generate first-level encryptions, that can be opened only by the intended recipient (Alice), or second-level ones that can be opened by any of the recipient's delegatees (e.g., Bob). Indeed, this property implies that even if Bob and the proxy collude, they will not be able to open any of Alice's first level-encryptions!

In general, collusion "safeness" allows Alice to 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., [41, 5, 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:* Dodis and Ivan [16] suggested applying generic key-insulation techniques [17, 14, 15] 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 Dodis and Ivan pointed out [16], one method for delegating decryption rights is to create a cryptosystem that has a two-stage decryption procedure with two different secret keys. In practice, Alice's secret key s is divided into two shares: s_1, given to the proxy, and s_2, given to Bob. A ciphertext intended for Alice can be partially decrypted by the proxy via s_1. Bob can complete the decryption process by using s_2 and then recover the message. We already noticed that this "secret sharing" approach does not exactly yield to proxy re-encryption schemes given that Bob must use secrets other than his own to recover the plaintext (i.e., there is no transformation of a ciphertext under Alice's public key into one under Bob's). In particular, this implies that the schemes as presented in [16] are not key optimal, proxy invisible, or collusion-"safe." Notice that there are trivial solutions to the collusion-"safe" problem when Alice is allowed to use two different key pairs, but we are interested in solutions that minimize the number of keys to safeguard and manage while remaining efficient. Indeed, in our first attempt, we try to improve on this by

providing a cryptosystem that generates ciphertexts that can be fully decrypted using either of two distinct keys. In particular, we consider a variant of the Paillier cryptosystem with two trapdoors proposed by Cramer and Shoup [13]. For simplicity, we will describe a version that is only semantically secure and we refer to the original work [13] for the full CCA2 secure scheme. This simplified scheme was described in [11] where the authors also show a variant of the scheme in [13] that works in the cyclic group of quadratic residues modulo n^2.

The public key is $(n, g, h = g^x)$ with g of order $\lambda(n) = 2p'q'$, the *master* secret key is the factorization of $n = pq$ (where $p = 2p' + 1, q = 2q' + 1$ are safe primes), and the "weak" secret key is $x \in [1, n^2/2]$. (As remarked in [13], such a g can be easily found by selecting a random $a \in Z^*_{n^2}$ and computing $g = -a^{2n}$.) To encrypt a message $m \in \mathbb{Z}_n$, select a random $r \in [1, n/4]$ and compute: $T_1 = g^r, T_2 = h^r(1 + mn) \pmod{n^2}$.

If x is known, then the message can be recovered as: $m = L(T_2/T_1^x \bmod n^2)$, where $L(u) = \frac{u-1}{n}$, for all $u \in \{u < n^2 \mid u = 1 \bmod n\}$. If (p, q) are known, then m can be recovered from T_2 by noticing that $T_2^{\lambda(n)} = g^{\lambda(n)xr}(1 + m\lambda(n)n) = (1 + m\lambda(n)n)$. Thus, given that $\gcd(\lambda(n), n) = 1$, m can be recovered as: $m = L(T_2^{\lambda(n)} \bmod n^2)[\lambda(n)]^{-1} \bmod n$.

Part of the cryptosystem above can be seen as a variation of Elgamal when working modulo a squared composite number. So, similarly to the Dodis-Ivan scheme, we can divide x into two shares x_1 and x_2, such that $x = x_1 + x_2$. The share x_1 is given to the proxy while x_2 is stored by Bob. The scheme is collusion-safe since only the "weak" secret x is exposed if Bob and the proxy collude, but the factors of n, p and q, remain secret. Indeed, one could send only the value T_2, rather than the ciphertext pair (T_1, T_2), to allow Alice, and only her, to decrypt the message. (Remember that we are assuming that ciphertexts are generated correctly.) Although collusion-"safe," this scheme is not key optimal or proxy invisible but it remains theoretically interesting because it is not based on bilinear pairings. However, it cannot yet be seen as a pure proxy re-encryption scheme since there is no transformation of ciphertexts computed under Alice's key into ones under Bob's.

One way to address this, which also applies to the Dodis-Ivan schemes, is to let the proxy store Bob's shares encrypted under his own public key. For instance, in the case where Alice is the delegator, the proxy could store x_1 and x_2, the latter encrypted under Bob's public key. The encrypted share will be sent to Bob along with the ciphertext partially decrypted by the proxy. This solution, however, is not satisfactory: It requires more bandwidth, it doubles the cost of decrypting, it forces Bob to perform distinct decryption procedures based on whether he receives ciphertexts intended for him or ciphertexts from the proxy, and it complicates the revocation of decryption rights.

A Second Attempt. To minimize a user's secret storage and thus become key optimal, we present the BBS [7], Elgamal based [19] scheme operating over two groups G_1, G_2 of prime order q with a bilinear map $e : G_1 \times G_1 \to G_2$. 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, D_2).** To decrypt a first-level ciphertext $c_a = (\alpha, \beta)$ with secret key $sk = a$, compute

$m = \beta/\alpha^{1/a}$. To decrypt a second-level ciphertext $c_a = (\alpha, \beta)$ with secret key $sk = a$, compute $m = \beta/e(\alpha, g)^{1/a}$.

Discussion of Scheme. This scheme is very attractive; it is unidirectional, non-interactive, proxy invisible, collusion-safe, key optimal, and non-transitive. In particular, notice that first-level encryptions intended for Alice are safe even if Bob and the proxy collude. Indeed, the weak secret $g^{1/a}$ cannot be used to decrypt first-level encryptions (but only second-level ones, which Bob and the proxy can open anyway).

The scheme is also very efficient since both encryption and decryption operations are similar to those of plain Elgamal while the pairing computation is only performed by the proxy.

The security of this scheme depends upon (at least) the assumption that the following problem is hard in (G_1, G_2):

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

To see where the above assumption comes into play, think of g^a as g^{bk} for some $k \in \mathbb{Z}_q$. Now, consider the second-level ciphertext $c = (g^a, mQ)$ encrypted for public key g^b for message m. If $Q = e(g,g)^{a/b} = e(g,g)^{bk/b} = e(g,g)^k$, then c is a proper encryption of m; otherwise, it is an encryption of some other message $m' \neq m$. Thus, an adversary that can break the above decisional assumption can easily be made into an adversary that breaks the semantic security of this scheme. Recently, the above assumption was proven hard in the generic group model by Dodis and Yampolskiy [18]. (Indeed, Dodis and Yampolskiy address a stronger version called q-Decisional Bilinear Diffie-Hellman Inversion (q-DBDHI) where for a random $g \in G_1$, $x \in \mathbb{Z}_q$, and $Q \in G_2$, given $(g, g^x, g^{x^2}, \ldots, g^{x^q}, Q)$, it is hard to decide if $Q = e(g,g)^{1/x}$ or not.)

However, the security of the above scheme also appears to rely on the assumption that given (g, g^a), the value a cannot be derived from seeing the a-th root of a polynomial set of random values. (This appears necessary to generate the appropriate re-encryption keys). Although this assumption seems plausible in a group of prime order, by making a few alterations to this core idea we are able to 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 $c_{a,2} = (Z^{a_2 k}, mZ^k)$).
- **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 $c_{a,r} = (g^k, mZ^{a_1 k})$.
- **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 $a_i \in sk_a$, compute $m = \beta/\alpha^{1/a_i}$ for $i \in \{1, 2\}$. To decrypt a second-level ciphertext $c_a = (\alpha, \beta)$ with secret key $a_1 \in sk_a$, compute $m = \beta/e(\alpha, g)^{a_1}$.

Discussion of Scheme. This scheme is similar to the previous one, except to accept delegations, a user must store two secret keys. If Bob and the proxy collude, they cannot decrypt first-level encryptions intended for Alice. Indeed, they can recover only the weak secret g^{a_1} that can only be used to decrypt second-level encryptions (which Bob and the proxy can already open anyway).

As in our previous scheme, both encryption and decryption operations are similar to those of plain Elgamal, thus very efficient, while the pairing computation is performed only by the proxy.

The security of this scheme relies on an extension of the decisional bilinear Diffie-Hellman (DBDH) assumption [8, 12]; the proof of Boneh and Franklin [8] that the DBDH problem is hard in generic groups, in the sense of Shoup [40], 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. When no delegations are made, original first-level ciphertexts of the form (Z^{a_1k}, mZ^k) are exactly like Elgamal [19] and thus their external security only depends on DDH in G_2.

Theorem 3.1 *The above scheme is correct and secure assuming the extended Decisional Bilinear Diffie-Hellman (eDBDH) 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).*

More precisely, any adversary that can break the standard security of this scheme with probability $(1/2+\varepsilon)$ can be used to break the eDBDH problem in (G_1, G_2) with probability $(1/2+\varepsilon/2)$. Any adversary that can break the master secret security of this scheme with probability ε can be used to break the discrete logarithm problem in G_1 with probability ε.

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 an E_1 ciphertext takes the form (Z^{a_1k}, mZ^k). This construction is equivalent to that of the form (Z^k, mZ^{a_1k}) [19]. Now, it is clear if the E_2 ciphertext of the form (g^k, mZ^{a_1k}) is secure, then so are the E_1 versions, since E_2 ciphertexts reveal strictly more information (i.e., $g^k \in G_1$). Thus, it suffices to argue the security of the E_2 ciphertexts only.

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

1. On eDBDH input $(y, y^a, y^b, y^c, e(y,y)^{bc^2}, e(y,y)^d)$, the simulator sets up a proxy re-encryption world for the adversary A with the goal of using A to decide if $d = abc$ or not. To begin, the simulator outputs the global parameters for the system (g, Z). Here, for reasons we will later see, the simulator sets $g = y^c$, $Z = e(g,g) = e(y,y)^{c^2}$. Next, the simulator sends to adversary A the target public key $pk_B = (e(y,y)^{bc^2} = Z^b, (y^c)^t = g^t)$, where t is randomly selected from $\mathbb{Z}_q$ by the simulator. Thus, we can think of (b,t) as the secret key of the target user.
2. Next, for $i = 1$ up to poly(k), A can request:
   (a) $rk_{x\to B}$, a delegation to B from a party corrupted by A. A can generate these delegations for as many corrupted users as it liked by internally by running $(pk_x, sk_x) \leftarrow KG(1^k)$ and computing $rk_{x\to B} = (g^t)^{sk_{(x,1)}}$, where $sk_x = (sk_{(x,1)}, sk_{(x,2)})$.
   (b) $rk_{B\to h}$, a delegation from B to an honest party h. The simulator randomly selects two values $r_{(h,1)}, r_{(h,2)} \leftarrow \mathbb{Z}_q$, sets $rk_{B\to h} = (y^b)^{r_{(h,2)}} = g^{b(r_{(h,2)}/c)}$ and $pk_h = (Z^{r_{(h,1)}}, y^{r_{(h,2)}} = g^{r_{(i,2)}/c})$, and sends $(pk_h, rk_{B\to h})$ to A. The corresponding secret key is $sk_h = (r_{(h,1)}, (r_{(h,2)}/c))$.
   (c) $rk_{h\to B}$, a delegation to B from an honest party h. The simulator uses either the recorded value $r_{(h,1)}$ from the previous step if the honest party already exists, or generates fresh random values for a new party, and computes $rk_{h\to B} = (g^t)^r_{(h,1)}$.

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 $s \in \{0,1\}$, computes the ciphertext $c_s = (y^a, m_s e(y,y)^d) = (g^{a/c}, m_s e(g,g)^{d/c^2})$, sends (c_s, τ) to A, and waits for A to output $s' \in \{0,1\}$.
4. If $s = s'$, then S guesses "$d = abc$"; otherwise S guesses "$d \neq abc$".

First, we observe that if $d = abc$, then the simulation is perfect; that is, the ciphertext output is of the proper form $(g^{a/c}, m_b Z^{(abc)/c^2} = m_b Z^{b(a/c)})$ for the user with $sk_{(B,1)} = b$. However, if $d \neq abc$, then m_b is information-theoretically hidden from A, since d was chosen independently of a, b, c. Thus, if A succeeds with probability $1/2+\varepsilon$, then S succeeds with probability $(1/2+\varepsilon)$ (when $d = abc$) and probability exactly $1/2$ (when $d \neq abc$), for an overall success probability of $(1/2+\varepsilon/2)$. This contradicts the eDBDH assumption when ε is non-negligible.

Master Secret Security. Suppose an adversary A can recover the secret key of a targeted user B (i.e., $sk_B = (sk_{(B,1)}, sk_{(B,2)})$) with non-negligible probability ε by interacting with B according to the second part of definition 2.3, then we can build an adversary S that takes discrete logs in G_1 with probability ε. Let us focus our attention on recovering only the value $sk_{(B,1)}$ (which is arguably the most valuable of the two). Our simulator S works as follows:

1. On input (g, g^a) in G_1, output the global parameters $(g, Z = e(g,g))$ and the target public key $pk_B = (e(g,g^a), g^{sk_{(B,2)}})$, where $sk_{(B,2)}$ is chosen at random from $\mathbb{Z}_q$. We can think of $sk_{(B,1)} = a$.
2. Next, for $i = 1$ up to poly(k), A can request:
   (a) $rk_{B\to x}$, a delegation from B to a party corrupted by A. S randomly selects $r_{(x,1)}, r_{(x,2)} \leftarrow \mathbb{Z}_q$, sets $rk_{B\to x} \leftarrow g^{ar_{(x,2)}}$, $pk_x = (Z^{r_{(x,1)}}, g^{r_{(x,2)}})$, and $sk_x = (r_{(x,1)}, r_{(x,2)})$, and sends $(pk_x, sk_x, rk_{B\to x})$ to A.
   (b) $rk_{x\to B}$, a delegation to B from a party corrupted by A. A can generate these delegations internally by running $(pk_x, sk_x) \leftarrow KG(1^k)$ and computing $rk_{x\to B} = (g^{sk_{(B,2)}})^{sk_{(x,1)}}$.
3. Eventually, A must output a purported secret key for B 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 B, despite accepting and providing numerous delegations to B, because otherwise, S can efficiently solve the discrete logarithm problem in G_1. □

3.2 Temporary Unidirectional Proxy Re-encryption

In this section, we improve our temporary unidirectional proxy re-encryption scheme over the conference version of this paper [4], by a slight alteration in the first-level encryption which does not increase the running time, but allows us to prove the scheme's security under a more standard assumption.

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 in such a way that it can only be decrypted by A, compute $Z_i^{a_r k} = e(g^{a_r}, h_i)^k$ and output $c_{a,r} = (Z_i^{a_r k}, mZ_i^k)$.
- **Second-Level Encryption (E_2).** To encrypt $m \in G_2$ under pk_a during time period i in such a way that it can be decrypted by A and her delegatees, compute $Z_i^{a_r k} = e(g^{a_r}, h_i)^k$, and 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 $Z_i^{b_i a_r k} = e(g^{a_0 k}, rk_{A\to B})$ 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, D_2).** To decrypt a first-level ciphertext $c_{a,j} = (\alpha, \beta)$ with secret key $a_j \in \{a_r, a_1, a_2, \ldots\}$ (corresponding to a re-encryption from the jth time period or a first-level original ciphertext with permanent key a_r), compute $m = \beta/\alpha^{1/a_j}$. To decrypt a second-level ciphertext $c_{a,j} = (\alpha, \beta)$ with secret key (a_0, a_r), compute $m = \beta^{a_0}/e(\alpha, h_j)^{a_r}$.

Discussion of Scheme. A single *global* change can invalidate all previous delegations without *any* user needing to change their public key.

Theorem 3.2 *The above scheme is correct and secure assuming the Decisional Bilinear Diffie-Hellman (DBDH) 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 $Q = e(g,g)^{abc}$ (standard security) and the discrete logarithm assumption (master secret security).*

More precisely, let T be the maximum number of time steps. Any adversary that can break the standard security of this scheme with probability $(1/2 + \varepsilon)$ can be used to break the DBDH problem in (G_1, G_2) with probability $(1/2 + \varepsilon/(2T))$. Any adversary that can break the master secret security of this scheme with probability ε can be used to break the discrete logarithm problem in G_1 with probability ε.

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 which produce different types of ciphertexts. Our security proof will address both styles of ciphertexts.

Standard Security. Let T be the maximum number of time periods. Suppose A distinguishes E_1 ciphertexts with non-negligible probability (we will address E_2 shortly), we simulate an adversary S that decides DBDH as follows:

1. On input (g, g^a, g^b, g^c, Z^d), the simulator sends A the global parameters $(g, e(g,g) = Z)$ and the target public key $pk_B = (g^t, g^a)$, where t is randomly selected from $\mathbb{Z}_q$, and the corresponding secret key is $sk_B = (t, a)$. The simulator also honestly generates and publishes the public keys of all other parties.
2. For $j = 1$ up to T time periods, the simulator publishes the public delegation parameter for that time period $h_j = g^{x_j}$, where x_j is randomly selected from $\mathbb{Z}_q$. The simulator also publishes the delegation acceptance value $D_{(U,j)} = h_j^{z_{(U,j)}}$ for all users U, including B, where $z_{(U,j)}$ is randomly selected from $\mathbb{Z}_q$.
   (a) Next, for $i = 1$ up to poly(k), A can request:
      i. $rk_{x\to B}$, a delegation to B from a party corrupted by A. A can generate these delegations internally by running $(pk_x, sk_x) \leftarrow KG(1^k)$, where $sk_x = (sk_{(x,0)}, sk_{(x,r)})$, and computing $rk_{x\to B} = D_{(B,j)}^{sk_{(x,r)}/sk_{(x,0)}}$.
      ii. $rk_{B\to h}$, a delegation from B to an honest party h with delegation acceptance value $D_{(h,j)}$ for time period j. S computes and sends $rk_{B\to h} \leftarrow (g^a)^{x_j z_{(h,j)}/t} = D_{(h,j)}^{a/t}$ to A.
      iii. $rk_{h\to B}$, a delegation to B from an honest party h with secret key $sk_h = (sk_{(h,0)}, sk_{(h,r)})$. S trivially computes $rk_{h\to B} = D_{(B,j)}^{sk_{(h,r)}/sk_{(h,0)}}$.

3. Eventually, during the last time period, 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 $s \in \{0,1\}$, computes the ciphertext $c_s = (Z^d, e(g^b, g^c)m_s)$, sends (c_s, τ) to A, and waits for A to output $s' \in \{0,1\}$.
4. If $s = s'$, then S guesses "$d = abc$"; otherwise S guesses "$d \neq abc$".

First, we observe that if $d = abc$, then the simulation is perfect; that is, the ciphertext output is of the proper form $(Z^{abc}, Z^{bc}m_s)$ for the user with $sk_{(B,r)} = a$. However, if $d \neq abc$, then m_s is information-theoretically hidden from A. Thus, if A succeeds with probability $1/2+\varepsilon$ at distinguishing E_1 ciphertexts, then S succeeds with probability $(1/2+\varepsilon)$ (when $d = abc$) and probability exactly 1/2 (when $d \neq abc$), for an overall probability of $(1/2+2/\varepsilon)$. This contradicts the DBDH assumption when ε is non-negligible.

Now, suppose that A distinguishes E_2 ciphertexts with non-negligible probability, we simulate a different adversary S that decides DBDH as follows:

1. On input (g, g^a, g^b, g^c, Z^d), the simulator sends A the global parameters $(y = g^c, e(y,y) = e(g,g)^{c^2})$ and the target public key $pk_B = (y^{1/c} = g, y^{a/c} = g^{a/c})$ and the corresponding secret key is $sk_B = (1/c, a/c)$. The simulator also honestly generates and publishes the public keys of all other parties.
2. For $j = 1$ up to T time periods, the simulator publishes the public delegation parameter for that time period $h_j = y^{x_j} = (g^c)^{x_j}$, where x_j is randomly selected from $\mathbb{Z}_q$. The simulator also publishes the delegation acceptance value $D_{(U,j)} = h_j^{z_{(U,j)}/c} = g^{x_j z_{(U,j)}}$ for all users U, including B, where $z_{(U,j)}$ is randomly selected from $\mathbb{Z}_q$. The simulator pretends to give the temporary secret $(z_{(U,j)}/c)$ to each honest party (it cannot actually do so, since it does not know the value $1/c$). These acceptance values are generated without the $1/c$ term for all corrupted users.
   (a) Next, for $i = 1$ up to poly(k), A can request:
      i. $rk_{x\to B}$, a delegation to B from a party corrupted by A. A can generate these delegations internally by running $(pk_x, sk_x) \leftarrow KG(1^k)$, where $sk_x = (sk_{(x,0)}, sk_{(x,r)})$, and computing $rk_{x\to B} = D_{(B,j)}^{sk_{(x,r)}/sk_{(x,0)}}$.
      ii. $rk_{B\to h}$, a delegation from B to an honest party h with delegation acceptance value $D_{(h,j)}$ for time period j. S computes and sends $rk_{B\to h} = D_{(h,j)}^{sk_{(B,r)}/sk_{(B,0)}} = y^{x_j z_{(h,j)}(a/c)/(1/c)} = (g^a)^{x_j z_{(h,j)}}$.
      iii. $rk_{h\to B}$, a delegation to B from an honest party h. S computes and sends $rk_{h\to B} = D_{(B,j)}^{sk_{(h,r)}/sk_{(h,0)}}$ to A.
3. Eventually, during the last time period, 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 $s \in \{0,1\}$, computes the ciphertext $c_s = (g^b, m_s(Z^d)^{x_j}) = (y^{b/c}, m_s e(y, h_j)^{d/c^2})$, sends (c_s, τ) to A, and waits for A to output $s' \in \{0,1\}$.
4. If $s = s'$, then S guesses "$d = abc$"; otherwise S guesses "$d \neq abc$".

Now, we observe that if $d = abc$, then the simulation is perfect; that is, the challenge ciphertext is of the proper form $(y^{sk_{(B,0)}b}, m_s e(y, h_j)^{sk_{(B,r)}b})$. However, if $d \neq abc$, then m_s is information-theoretically hidden from A. Thus, if A succeeds with probability $1/2+\varepsilon$ at distinguishing E_2 ciphertexts, then S succeeds with probability $(1/2+\varepsilon)$ (when $d = abc$) and probability exactly 1/2 (when $d \neq abc$), for an overall probability of $(1/2+2/\varepsilon)$. This contradicts the DBDH assumption when ε is non-negligible.

Master Secret Security. Let T be the maximum number of time periods. Suppose an adversary A can recover the secret key of a targeted user B (i.e., $sk_B = (sk_{(B,1)}, sk_{(B,2)})$) with non-negligible probability by interacting with B according to the second part of definition 2.3, then we can build an adversary S that takes discrete logs in G_1. Let us focus our attention on recovering only the value $sk_{(B,2)}$. Our simulator S works as follows:

1. On input (g, g^a) in G_1, output the global parameters $(g, Z = e(g,g))$ and the target public key $pk_B = (g^{sk_{(B,0)}}, g^a)$, where $sk_{(B,0)}$ is chosen randomly from $\mathbb{Z}_q$. We can think of $sk_{(B,r)} = a$.
2. For $j = 1$ up to T time periods, the simulator publishes the public delegation parameter for that time period $h_j = g^{x_j}$, where x_j is randomly selected from $\mathbb{Z}_q$. The simulator also publishes the delegation acceptance value $h_j^{z_{(U,j)}}$ for all users U, including B, where $z_{(U,j)}$ is randomly selected from $\mathbb{Z}_q$.
3. Next, for $i = 1$ up to $\text{poly}(k)$, A can request:
   (a) $rk_{B\to U}$, a delegation from B to a party corrupted by A. Let j be the current time period. S sets $rk_{B\to U} = (g^a)^{x_j z_{(U,j)}/sk_{(B,0)}} = h_j^{az_{(U,j)}/sk_{(B,0)}} = h_j^{sk_{(B,r)} z_{(U,j)}/sk_{(B,0)}}$.
   (b) $rk_{U\to B}$, a delegation to B from a party corrupted by A. A can generate these delegations internally using the public information of B.
4. Eventually, A must output a purported secret key for B of the form (α, β). The simulator returns the value β.

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

4 Encrypted File Storage

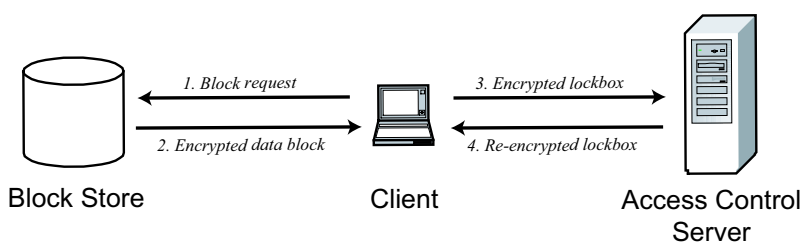

Figure 1: Operation of the proxy re-encryption file system. The user's client machine fetches encrypted blocks from the block store. Each block includes a lockbox encrypted under a master public key. The client then transmits lockboxes to the access control server for re-encryption under the user's public 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.

In this section we describe a file system which 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, our implementation represents the first experimental implementation and evaluation of a system using proxy re-encryption.

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 content keys used to encrypt files are themselves securely encrypted under a master public key, using a unidirectional proxy re-encryption scheme of the form described in this work. Because the access control server does not possess the corresponding secret key, it cannot be corrupted so as to gain access to the content keys necessary to 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 re-encryption 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* and *non-interactive*, meaning that users do not need to communicate or reveal 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 [22], a confidentiality-enabled version of the SFS read-only file system [23]. 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. We chose the Chefs architecture because it allowed us to experiment with different granularities of encryption (per-file and per-block) while providing a transparent file system interface for our experiments.

Chefs tags each content block with a lockbox. In the original Chefs design, the lockbox contains a 128-bit AES key, itself encrypted with a shared group AES key. Chefs assumes an out-of-band mechanism for content owners to distribute group keys to users.

4.1 Design and Implementation

To implement our proposed design, 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 public key using the "Third Attempt" bilinear Elgamal 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 parameters of the proxy re-encryption scheme causes a sealed lockbox to expand to several hundred bits, even though the underlying plaintext is 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 from the master key to the client's key, it performs the appropriate proxy re-encryption and returns the resulting ciphertext to the client, which can then decrypt the lockbox under its own secret key. 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 end-to-end 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 end-to-end 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 Gbyte RAM and an IBM 7200 RPM, 40 Gbyte, Ultra ATA/100 hard drive. The server machine was an Intel Pentium 4 2.8 GHz with 1 Gbyte RAM and a Seagate Barracuda 7200 RPM, 160 Gbyte, Ultra ATA/100 hard drive. Both systems were running Debian testing/unstable with the Linux 2.6.8 kernel. 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 msec, and the maximum sustained throughput of the network link at 7 Mbit/sec. We implemented the cryptographic primitives for the "Third Attempt" bilinear Elgamal scheme using version 4.83 of the MIRACL cryptographic library [39], which contains efficient implementations of the Tate pairing as well as fast modular exponentiation and point multiplication.

Cryptographic Benchmark. Table 2 presents average times over 100 runs of the cryptographic operations in the bilinear proxy re-encryption scheme (the third one from Section 3.1). The measurements 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 operations in our file system.

We were surprised that our 1.8 GHz AMD Athlon 2100 performed better than our 2.8 GHz Intel Pentium 4 server in the microbenchmarks. We attribute this advantage to modular arithmetic routines in MIRACL that perform faster on the Athlon. The MIRACL library provides many hints for selecting assembly code optimizations. Because other benchmarks such as the OpenSSL RSA "speed" test run faster on our

Parameter size	*Machine*	*Encryption*	*Decryption (by original recipient)*	*Re-encryption*	*Decryption (by delegatee)*
256-bit	client	3.1 msec	8.7 msec	8.6 msec	1.5 msec
	server	3.3 msec	8.8 msec	8.7 msec	1.5 msec
512-bit	client	7.8 msec	22.5 msec	22.0 msec	3.4 msec
	server	9.3 msec	26.5 msec	26.7 msec	4.1 msec

Table 2: Average operation times for 100 runs of the "Third Attempt" bilinear Elgamal proxy re-encryption scheme on our client and server. All operations refer to re-encryptable "second-level" ciphertexts.

server, we suspect that the Intel server would perform better with proper selection of optimizations in MIRACL.

We conducted our remaining benchmarks using various encryption granularities, including per-block and per-file. For each measurement, we report the median result of five samples. In all measurements, the server has a warm block cache and the client has a cold block cache. Our microbenchmarks, presented in Figures 2 and 3, include runs of the small-file and large-file test from the LFS suite of file system performance tests [37]. We use the read phases of the LFS test to measure the fundamental performance of our system.

The first test reads several small files. The second test consists of a sequential read of a large file. 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 content keys. 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 workload 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 44 seconds longer than Chefs. The 44 seconds corresponds exactly to the 13 msec round-trip time, 26.7 msec re-encryption time on the server, and 3.4 msec delegatee decryption time on the client for each of the 1,011 re-encryption RPCs (See Table 2).

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 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 Mbyte 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

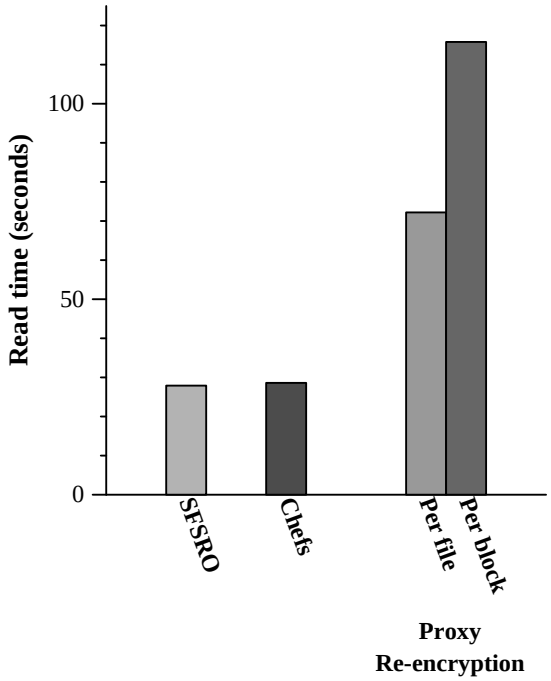

Figure 2: Small-file microbenchmark from LFS suite. We perform a complete read on 1,000 1 Kbyte files dispersed in 10 directories.

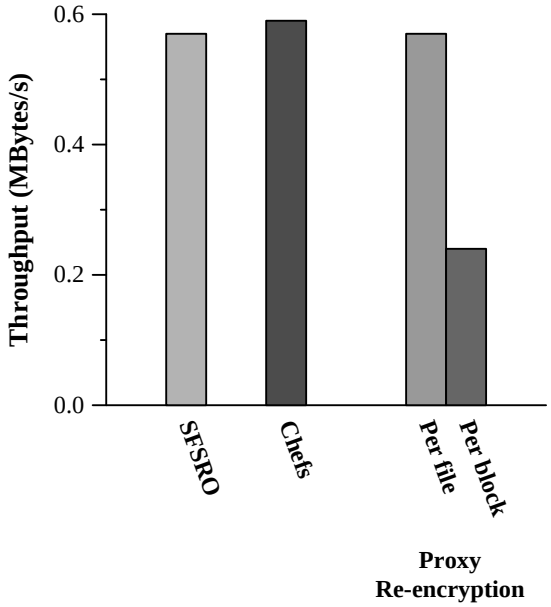

Figure 3: Large-file microbenchmark from LFS suite. We perform a sequential read on a 40 Mbyte file in 8 Kbyte blocks.

and Chefs experiments, the 7 Mbit bandwidth largely dominates the throughput.

The measurement of Chefs demonstrates that the cost of adding confidentiality to SFSRO comes at no cost to throughput when fetching data across the Internet. Chefs runs slightly faster than SFSRO, initially causing us to question the validity of the measurements. We conducted further measurements to ensure confidence in our results. We verified that both SFSRO and Chefs issue the same type and number of RPCs. The responses to content fetching RPCs in SFSRO are 8,200 bytes. The responses in Chefs are 8,244 bytes because of PKCS#7 padding overhead with AES-CBC encryption. We suspect that byte alignment and fortuitous interleaving of asynchronous RPCs allowed Chefs to slightly outperform SFSRO. In a run of the large-file benchmark, SFSRO encountered locking contention 16,054 times (when a callback encounters a locked a cache). Chefs encountered only slightly less locking contention — 15,981 times.

Because the large-file workload involves only a single file, the per-file proxy re-encryption has no discernible 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 13 msec network round-trip time.

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.

Note that the large file benchmark in this extended document differs from that of previous versions of this paper. Our original file system clients unintentionally prefetched file content. Furthermore, a race condition had the potential to generate unnecessary RPCs to fetch content. The race condition did not affect measurements in the original paper; the correct function always won the race. In later unpublished measurements, the race generated twice as many RPCs as necessary to fetch content. Removing the unintentional prefetching and fixing the race condition with a locking protocol slightly reduced the baseline throughput for sequential reads.

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. We first run `configure`, then compile with `make`. This CPU-intensive 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. The original paper did not take into account the time to run `configure`. Therefore, the new timings are slightly longer for all tests.

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.

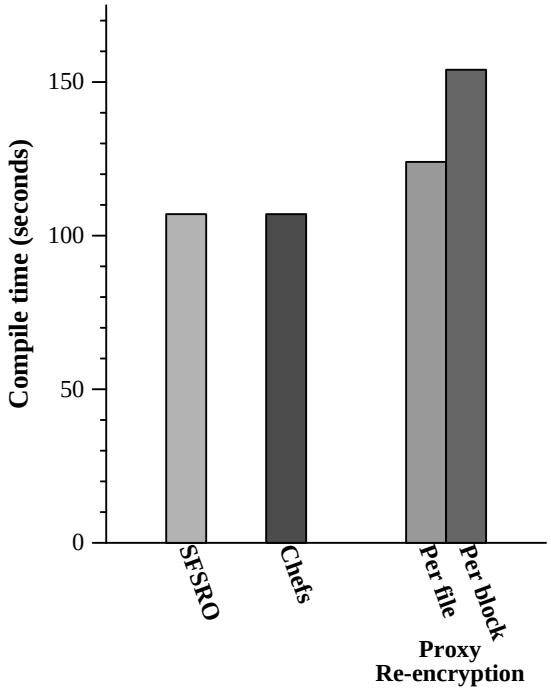

Figure 4: Application-level benchmark. We record the time to compile Emacs version 21.3.

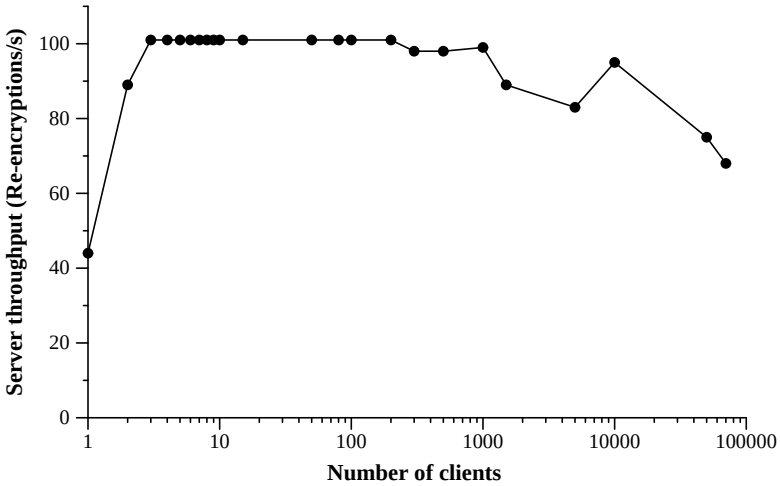

Figure 5: Aggregate access control server throughput. The server can tolerate 1,000 simultaneous re-encryption requests before showing signs of saturation.

4.3 Discussion

Our access control server acts like a credentials download service. For instance, PDM [35] 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 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 [33].

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 using a centralized, 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 and Future Work

In this paper, we explored proxy re-encryption from both a theoretical and practical 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 Chefs 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. Another challenging problem is to find unidirectional re-encryption schemes that allow ciphertexts to be re-encrypted in sequence and multiple times. 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.

As future work, we plan to explore definitions of proxy re-encryption to achieve CCA2 security in a

multi-user setting. This requires careful consideration of the secrets involved, including those held by the proxy and delegatees themselves. One promising direction is to deploy a variation of certain standard transformations, such as the one proposed by Fujisaki-Okamoto in [24], that allow to transform the ciphertext so that the non-malleability of the encrypted message is guaranteed while allowing the public key of the intended recipient to be changed.

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, Russ Cox, Eu-Jin Goh, Frans Kaashoek, Mahesh Kallahalla, Ronald L. Rivest, and the anonymous reviewers for helpful comments and discussions. 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 Act of 1996 (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] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In *Proceedings of Eurocrypt '02*, volume 2332 of LNCS, pages 83–107, 2002.

[4] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *Proceedings of the 12th Annual Network and Distributed System Security Symposium*, pages 29–44. Internet Society, February 2005.

[5] Joonsang Baek, Ron Steinfeld, and Yuliang Zheng. Formal proofs for the security of signcryption. In *Proceedings of Public Key Cryptography '02*, volume 2274 of LNCS, pages 80–98, 2002.

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

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

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

[9] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures. In *Proceedings of Eurocrypt '03*, volume 2656 of LNCS, pages 416–432, 2003.

[10] Dan Boneh, Hovav Shacham, and Ben Lynn. Short signatures from the Weil pairing. In *Proceedings of Asiacrypt '01*, volume 2248, pages 514–532, 2001.

[11] E. Bresson, D. Catalano, and D. Pointcheval. A simple public-key cryptosystem. In *Proceedings of Asiacrypt '03*, volume 2894 of LNCS, pages 37–54, 2003.

[12] Jung Hee Cheon and Dong Hoon Lee. Diffie-Hellman problems and bilinear maps. *Cryptology ePrint Archive: Report 2002/117*, 2001.

[13] Ronald Cramer and Victor 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.

[14] Yevgeniy Dodis, Matthew K. Franklin, Jonathan Katz, Atsuko Miyaji, and Moti Yung. Intrusion-resilient public-key encryption. In *Proceedings of CT-RSA '03*, volume 2612 of LNCS, pages 19–32, 2003.

[15] Yevgeniy Dodis, Matthew K. Franklin, Jonathan Katz, Atsuko Miyaji, and Moti Yung. A generic construction for intrusion-resilient public-key encryption. In *Proceedings of CT-RSA '04*, volume 2964 of LNCS, pages 81–98, 2004.

[16] Yevgeniy Dodis and Anca Ivan. Proxy cryptography revisited. In *Proceedings of the Tenth Network and Distributed System Security Symposium*, February 2003.

[17] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Key-insulated public key cryptosystems. In *Proceedings of Eurocrypt '02*, volume 2332 of LNCS, pages 65–82, 2002.

[18] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs an keys. In *Public Key Cryptography*, volume 3386 of LNCS, pages 416–431, 2005.

[19] Taher Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of Crypto '84*, pages 10–18, 1984.

[20] Amos Fiat and Adi 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.

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

[22] Kevin Fu. *Integrity and access control in untrusted content distribution networks*. PhD thesis, Massachusetts Institute of Technology, sept 2005.

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

[24] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Proceedings of Crypto '99*, volume 1666 of LNCS, pages 537–554, 1999.

[25] Steven D. Galbraith, Keith Harrison, and David Soldera. Implementing the Tate pairing. In *Proceedings of the Algorithmic Number Theory Symposium*, volume 2369 of LNCS, pages 324–337, 2002.

[26] Eu-Jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. SiRiUS: Securing remote untrusted storage. In *Proceedings of the Tenth Network and Distributed System Security Symposium*, pages 131–145, February 2003.

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

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

[29] Anthony Harrington and Christian 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.

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

[31] Antoine 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.

[32] 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.

[33] Jinyuan Li, Maxwell N. Krohn, David Mazières, and Dennis 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.

[34] Masahiro Mambo and Eiji Okamoto. Proxy cryptosystems: Delegation of the power to decrypt ciphertexts. *IEICE Trans. Fund. Electronics Communications and Computer Science*, E80-A/1:54–63, 1997.

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

[36] David Reed and Liba 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.

[37] 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.

[38] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptography*, 4:161–174, 1991.

[39] Michael Scott. MIRACL library, 2005. Indigo Software. http://indigo.ie/∼mscott/#download.

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

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

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