Proxy Re-Signatures: New Definitions, Algorithms, and Applications*

Giuseppe Ateniese† Susan Hohenberger‡

November 28, 2005

Abstract

In 1998, Blaze, Bleumer, and Strauss (BBS) proposed *proxy re-signatures*, in which a semi-trusted proxy acts as a *translator* between Alice and Bob. To translate, the proxy converts a signature from Alice into a signature from Bob on the same message. The proxy, however, does not learn any signing key and cannot sign arbitrary messages on behalf of either Alice or Bob. Since the BBS proposal, the proxy re-signature primitive has been largely ignored, but we show that it is a very useful tool for sharing web certificates, forming weak group signatures, and authenticating a network path.

We begin our results by formalizing the definition of security for a proxy re-signature. We next substantiate the need for improved schemes by pointing out certain weaknesses of the original BBS proxy re-signature scheme which make it unfit for most practical applications. We then present two secure proxy re-signature schemes based on bilinear maps. Our first scheme relies on the Computational Diffie-Hellman (CDH) assumption; here the proxy can translate from Alice to Bob and vice-versa. Our second scheme relies on the CDH and 2-Discrete Logarithm (2-DL) assumptions and achieves a stronger security guarantee – the proxy is only able to translate in one direction. Constructing such a scheme has been an open problem since proposed by BBS in 1998. Furthermore in this second scheme, even if the delegator and the proxy collude, they cannot sign on behalf of the delegatee. Both schemes are efficient and secure in the random oracle model.

1 Introduction

In a *proxy re-signature scheme*, a semi-trusted proxy is given some information which allows it to transform Alice's signature on a message m into Bob's signature on m, but the proxy cannot, on its own, generate signatures for either Alice or Bob.

This primitive was introduced at Eurocrypt '98 by Blaze, Bleumer, and Strauss [5] and yet very little follow up work has been done, to our knowledge. One explanation is that the BBS original construction [5] is inefficient and with limited features. Moreover, the definition of proxy re-signature in the BBS paper [5] is informal and has created some confusion between the notion

*This is the full version of a paper in Catherine Meadows and Paul Syverson, editors, *proceedings of the 12th ACM Conference on Computer and Communications Security* (CCS), pages 310-319, 2005.

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

‡Computer Science and Artificial Intelligence Laboratory; Massachusetts Institute of Technology; 32 Vassar Street; Cambridge, MA 02139, USA. Email: srhohen@mit.edu.

of proxy re-signature and the distinct one of *proxy signature* as introduced by Mambo, Usuda, and Okamoto [18].

Proxy signatures [18] allow Alice to delegate her signing rights to Bob but only if the proxy cooperates. In practice, Bob and the proxy can jointly generate a signature on arbitrary messages on Alice's behalf. This is usually accomplished by dividing Alice's secret into two shares which are distributed to Bob and the proxy (each gets only one share). A signature from Alice on a message is generated by combining two partial signatures on the same message computed by Bob and the proxy under their own shares, respectively.

In proxy re-signature [5], a proxy "translates" a perfectly-valid and publicly-verifiable signature from Alice on a certain message, $\sigma_A(m)$, into one from Bob on the same message, $\sigma_B(m)$. Notice that, in proxy re-signature, the two signatures, one from Alice and the other from Bob as generated by the proxy, can coexist and both can be publicly verified as being two signatures from two distinct people on the same message. Moreover, the proxy can convert a single signature into multiple signatures of several and distinct signers, and vice-versa!

Any proxy re-signature can be used to build a proxy signature, but the reverse is not necessarily true. For instance, it is possible to build a proxy signature based on RSA (as in [12], by splitting Alice's secret d into d_1 and d_2 such that $d = d_1 + d_2$) but it is not possible to have a proxy re-signature scheme that translates between two publicly-verifiable RSA signatures sharing the same modulus (because of the *common modulus* problem/attack). Another unique property of proxy re-signature is that the "translation" from one signature to another can be performed in sequence and multiple times by distinct proxies without even requiring the intervention of the signing entities. Thus, the valuable secret keys can remain *offline*. The signatures generated in this process are all valid and publicly-verifiable signatures on the same message from distinct entities.

We re-open the discussion of proxy re-signature by providing four separate results: (1) we first motivate the need for improved schemes, by pointing out that the original BBS scheme [5], while satisfying their security notion, is unsuitable for most practical applications, including the ones proposed in the original paper, (2) we provide formal definitions and a security model, (3) we introduce provably secure proxy re-signature constructions from bilinear maps, and (4) we present new applications and perform comparisons with other cryptographic primitives. Incidentally, we also introduce a new signature scheme based on bilinear maps where two signing secrets are associated to a single public key. Let us briefly describe these results in more detail.

Remarks on the BBS Scheme. The authors of the BBS paper introduced several potential applications of proxy re-signatures. By taking a careful look at how one might wish to use proxy re-signatures in practice, we noticed that the BBS construction [5] has inherent limitations. In short, their scheme is actually "proxy-less" since it is possible to recover the information that would be stored at the proxy (the re-signature key) by looking at the original signature and its transformation. This precludes the possibility of having a proxy in the first place since anyone would be able to impersonate the proxy itself once a *single* re-signature is released. Moreover, the BBS scheme is what the authors called *"symmetric"* [5], which means that, from the re-signature key (which is public!), Alice can recover Bob's secret key or vice-versa.

Finding secure schemes without these limitations turned out to be a very difficult task. Many proxy re-signature schemes based on standard signature algorithms that we investigated were found susceptible to the same type of problems. Several pairing-based signature schemes also failed or it was not clear how to turn them in proxy re-signatures.

Formal Definitions and Security Model. There is no formal definition of proxy re-signature in the BBS paper [5] and their security guarantee holds for only a few limited applications, given the problems we mentioned earlier.

For this reason, we formalize the notion of a proxy re-signature and provide a security model that incorporates the desirable property of having the re-signature key safely stored at the proxy (so that it is reasonable to talk about *proxies* in this context). First, this allows us to make meaningful claims about our schemes' security. In particular, in Section 3.3, we realize a strong notion of security: Suppose Alice delegates to Bob, via a proxy, the ability to change his signatures into hers. As one might expect, even when the proxy and Alice collude, they cannot recover any information about Bob except his public key. More surprisingly, we show that even when the proxy and Bob collude, they can only recover a weak version of Alice's secret key – that only gives them the power to compute what Alice had already delegated to them. Secondly, our formal notion allows us to view the primitive abstractly, for easier reasoning about its applications in Section 4.

Two Proxy Re-Signature Constructions. We present two different constructions based on bilinear maps. The first is a *bidirectional* proxy re-signature in which the proxy can translate from Alice to Bob and vice-versa using a single proxy key. The scheme is very attractive for its simplicity. Unlike the bidirectional BBS scheme, here the proxy can keep his proxy keys private. This scheme also allows for *multi-use*, meaning that a signature may be transformed from Alice to Bob, then from Bob to Carol, and so on. The security of the scheme is based on the Computational Diffie-Hellman (CDH) assumption, i.e., given (g, g^x, g^y), it is hard to compute g^{xy}, in the random oracle model.

The second scheme is *unidirectional* in that the proxy can be given information that allows it to translate from Alice to Bob, but not from Bob to Alice. This is the first construction of a unidirectional scheme since it was proposed as an open problem by BBS seven years ago [5]. (In BBS, they refer to such schemes as *asymmetric*.) Here, we allow the re-signature key to be public so that anyone can act like a proxy but, at the same time, we ensure that certain important security properties are guaranteed based on its unidirectional nature. (We also provide some insight on how one might keep this proxy key private.) The security of this scheme is based on the CDH and 2-Discrete Logarithm (2-DL), i.e., given (g, g^x, g^{x^2}), it is hard to compute x, assumptions in the random oracle model.

Finally, we note that our unidirectional scheme introduces a new signature algorithm that may be of independent interest, because it allows the signer to use *two secrets* (strong and weak) for a *single* public key (more details in Section 3.4).

Applications. We propose exciting new applications of proxy re-signatures in Section 4. In particular we show how to use proxy re-signatures to provide a proof that a certain path in a graph was taken. In the simplest case, the basic idea is that each node in the path (except the first) is only given a re-signature key which allows it to translate signatures from adjacent nodes, but *not* a signing key. For instance, given three nodes in a path $A \rightarrow B \rightarrow C$, we give the first node A's signing key, while the second node is only able to translate signatures from A into signatures from B, without using (or storing) B's signing key. Then, node C only has to verify a single signature from B even if several nodes (not just A) precede B in the path.

Our technique requires some pre-configuration but provides several benefits: (1) all (but the first) nodes in the graph do not store any signing key so that, even if a node is compromised, no new messages can be injected in the graph, (2) only a single signature must traverse the path, i.e., there

is no need to accumulate signatures and public keys while traversing the path, (3) no information on the path itself is collected so that, if applications require it, the actual path traversed by a message could remain *private*, i.e., the last node of the path knows that the message followed a legitimate path but does not know which one (we stress that this property is optional).

We will also introduce other interesting applications. For instance, we show how to use proxy re-signatures to share existing public-key certificates and save the cost of obtaining and distributing new ones. In particular, a signature $\sigma'(m)$, that can only be verified with a new and/or uncertified public key P', could be transformed into $\sigma(m)$ that can be verified with a trusted and certified public key P. Depending on the application, translating between signatures could be more convenient then requesting a certificate for P' and distributing it to several parties.

A certificate itself is often implemented as a digital signature from a certification authority (CA). There are cases where certificates released by CA1 cannot be verified by entities trusting only certificates released by CA2. A proxy could be set up to transform certificates from CA1 into ones from CA2, effectively removing the need for building complex hierarchies between certification authorities (at least temporarily).

We also show how to generate anonymous signatures that, much like group signatures, can be used to hide a group's membership while providing accountability. Indeed, a proxy could translate signatures from group members into signatures from the group manager. The advantage here is that the proxy does not store any signing key and the key of the group manager cannot get exposed even if the proxy is compromised.

1.1 Related Work

Proxy re-signatures [5] should not be confused with the similar sounding *proxy signatures* [18, 12] as previously discussed. In particular, definitions in Dodis et al. [12] are for proxy signatures: In their general construction, Bob's signature is seen as a *double signature* which includes a signature from Alice and one from the proxy. There is clearly no translation between valid Alice's signatures into Bob's ones and Alice's signing secret key is actually provided to her by Bob (or by a trusted authority).

Proxy re-signatures share some properties with the transitive signatures as introduced by Micali and Rivest [20] and extended by Bellare and Neven [3]. In a transitive signature scheme, a single master signer signs edges of a graph in such a way that *anyone* (i.e., not only a proxy) in possession of signatures on edges ab and bc can compute the master's signature on edge ac. There is a symmetry between these schemes: in a transitive signature, the verification key stays constant while the message content changes, while in a proxy re-signature scheme the message content remains constant while the verification key changes. Transitive signatures may also be appropriate for *some* of our authenticated routing applications in Section 4.

Our work is also related to multi and aggregate signatures. Multi-signatures [19, 6, 22, 21] allow n users, each with a public and private key pair (pk_i, sk_i), to all sign the same message m and have the result be just one single signature. Similarly, aggregate signatures, introduced in [8], allow each of the n users to sign a distinct message such that the computed n signatures, on n distinct messages, can be aggregated into a single one. Our multi-use proxy re-signatures can be used as an alternative to multi-signatures, but not to aggregate signatures, given that we allow transformations to be made only between signatures on the same message. In certain applications (e.g., those requiring a chain of signers), multi and aggregate signatures have a potential drawback: the verifier must possess and trust n public keys in order to verify the final *compressed* signature. In

Section 4, we discuss how, when proxy re-signatures are used instead, the verifier need only possess and trust *one* public key. This cuts down on the storage needs and key distribution headaches of some practical applications.

We notice that our first proxy re-signature scheme, in Section 3.3, uses a construction similar to Boldyreva's multi-signature [6] and Dodis and Reyzin's "verifiably committed signature" [13] constructions which are both based on the short signature of Boneh, Lynn, and Shacham [10, 9]. However, the purpose, definition, and the security model of these cryptographic primitives are different and unrelated.

2 Definitions

Our protocols are based on bilinear maps [7, 10, 16], which can be efficiently implemented [14].

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, $\hat{G}_1$, 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-signature scheme.

Definition 2.2 (Proxy Re-Signature) *A* proxy re-signature scheme *is a tuple of (possibly probabilistic) polynomial time algorithms* (KeyGen, ReKey, Sign, ReSign, Verify)*, where:*

- ($\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}$) form the standard key generation, signing, and verification algorithms.
- On input $(pk_A, sk_A^*, pk_B, sk_B)$, the re-encryption key generation algorithm, ReKey, outputs a key $rk_{A \rightarrow B}$ for the proxy. (Note: $rk_{A \rightarrow B}$ allows to transform A's signatures into B's signatures – thus B is the delegator.) The input marked with a '$*$' is optional.
- On input $rk_{A \rightarrow B}$, a public key pk_A, a signature σ, and a message m, the re-signature function, ReSign, outputs $\sigma_B(m)$ if $\mathsf{Verify}(pk_A, m, \sigma)$ and $\perp$ otherwise.

Correctness. The correctness property has two requirements. For any message m in the message space and any key pairs $(pk, sk), (pk', sk') \leftarrow \mathsf{KeyGen}(1^k)$, let $\sigma = \mathsf{Sign}(sk, m)$ and $rk \leftarrow \mathsf{ReKey}(pk, sk, pk'\, sk')$. Then the following two conditions must hold:

$$\mathsf{Verify}(pk, m, \sigma) = 1 \quad \text{and} \quad \mathsf{Verify}(pk', m, \mathsf{ReSign}(rk, \sigma)) = 1.$$

That is, all signatures validly formed by either the signing or re-signing algorithms will pass verification.

Internal and External Security. Our security model protects users from two types of attacks: those launched from parties outside the system (*External Security*), and those launched from parties inside the system, such as the proxy, another delegation partner, or some collusion between them (*Internal Security*). We now provide both intuition and a formalization of these security notions.

External Security: Our first security notion protects a user from adversaries outside the system (i.e., excluding the proxy and any delegation partners). This is the proxy equivalent of *strong* existential unforgeability under adaptive chosen-message attack (where an adversary cannot create a new signature even for a previously signed message) [1]. For some applications, it may also make sense to require only the standard notion of existential unforgeability (where a forgery must be on a new message) [15], although our constructions are able to satisfy this stronger notion.

Formally, for any non-zero $n \in \text{poly}(k)$ and all PPT algorithms $\mathcal{A}$,

$$\Pr[\{(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(1^k)\}_{i\in[1,n]}, \\ (t, m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}_{sign}(\cdot,\cdot),\mathcal{O}_{resign}(\cdot,\cdot,\cdot,\cdot)}(\{pk_i\}_{i\in[1,n]}) : \\ \mathsf{Verify}(pk_t, m, \sigma) = 1 \wedge (1 \le t \le n) \wedge (t, m, \sigma) \notin Q] < 1/\text{poly}(k)$$

where the oracle $\mathcal{O}_{sign}$ takes as input an index $1 \le j \le n$ and a message $m \in M$, and produces the output of $\mathsf{Sign}(sk_j, m)$; the oracle $\mathcal{O}_{resign}$ takes as input two distinct indices $1 \le i, j \le n$, a message m, and a signature σ, and produces the output of $\mathsf{ReSign}(\mathsf{ReKey}(pk_i, sk_i, pk_j, sk_j), pk_i, \sigma, m)$; and Q denotes the set of (index, message, signature) tuples (t, m, σ) where $\mathcal{A}$ obtained a signature σ on m under public key pk_t by querying $\mathcal{O}_{sign}$ on (t, m) or $\mathcal{O}_{resign}(\cdot, t, m, \cdot)$.

In the above security notion, the proxy is *required* to keep the re-signature keys private (or it is easy for an adversary to "win".). For some unidirectional schemes, however, one might want these values to be public (i.e., making all users proxies). When this is the case, there are no "external adversaries" to the system, and thus we look instead to the internal security guarantee below.

Internal Security: Our second security notion protects a user, as much as possible, when they are fooled into trusting a rogue proxy and/or delegation partner (who may be colluding). Intuitively, there are three guarantees to make.

1. Limited Proxy: If the delegator and the delegatee are both honest, then: (1) the proxy cannot produce signatures for the delegator unless the message was first signed by one of her delegatees, and (2) the proxy cannot create *any* signatures for the delegatee. This is identical to the external security game, except that instead of a re-signing oracle $\mathcal{O}_{sign}$, $\mathcal{A}$ may directly obtain the re-signature keys via $\mathcal{O}_{rekey}$.

Unidirectional: For any non-zero $n \in \text{poly}(k)$ and all PPT algorithms $\mathcal{A}$,

$$\Pr[\{(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(1^k)\}_{i\in[1,n]}, \\ (t, m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}_{sign}(\cdot,\cdot),\mathcal{O}_{rekey}(\cdot,\cdot)}(\{pk_i\}_{i\in[1,n]}) : \\ \mathsf{Verify}(pk_t, m, \sigma) = 1 \wedge (1 \le t \le n) \wedge (t, m) \notin Q] < 1/\text{poly}(k)$$

where the oracle $\mathcal{O}_{rekey}$ takes as input two distinct indices $1 \le i, j \le n$ and returns the output of $\mathsf{ReKey}(pk_i, sk_i, pk_j, sk_j)$; and Q denotes the set of pairs (t, m) where $\mathcal{A}$ obtained a signature on m under public key pk_t or one of its delegatee key's by querying $\mathcal{O}_{sign}$.

Bidirectional: Since both parties mutually delegate, the set Q includes all messages associated with *any* $\mathcal{O}_{sign}$ query.

2. Delegatee Security: If the delegatee is honest, then he is "safe" from a colluding delegator and proxy. That is, they cannot produce any signatures on his behalf. We associate the index 0 to the delegatee.

Undirectional: For any non-zero $n \in \text{poly}(k)$ and all PPT algorithms $\mathcal{A}$,

$$\Pr[\{(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(1^k)\}_{i\in[0,n]}, (m,\sigma) \leftarrow \mathcal{A}^{\mathcal{O}_{sign}(0,\cdot),\mathcal{O}_{rekey}(\cdot,\star)}(pk_0, \{pk_i, sk_i\}_{i\in[1,n]}) : \mathsf{Verify}(pk_0, m, \sigma) = 1 \wedge (m,\sigma) \notin Q] < 1/\text{poly}(k)$$

where $\star \neq 0$ and Q is the set of pairs (m, σ) such that $\mathcal{A}$ queried $\mathcal{O}_{sign}(0, m)$ and obtained σ.

Bidirectional: Since both parties are delegators, this property does not apply.

3. Delegator Security: If the delegator is honest, then she is "safe" from a colluding delegatee and proxy. That is, there are distinguishable signatures for a user based on whether she used her strong secret key or her weak secret key. The colluding delegatee and proxy cannot produce *strong* (a.k.a., first-level) signatures on her behalf. We associate the index 0 to the delegator.

Unidirectional: For any non-zero $n \in \text{poly}(k)$ and all PPT algorithms $\mathcal{A}$,

$$\Pr[\{(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(1^k)\}_{i\in[1,n]}, (m,\sigma) \leftarrow \mathcal{A}^{\mathcal{O}_{sign}(0,\cdot),\mathcal{O}_{rekey}(\cdot,\cdot)}(pk_0, \{pk_i, sk_i\}_{i\in[1,n]}) : \mathsf{Verify}(pk_0, m, \sigma) = 1 \wedge (m,\sigma) \notin Q] < 1/\text{poly}(k)$$

where σ is a first-level signature, and Q is the set of pairs (m, σ) where $\mathcal{A}$ queried $\mathcal{O}_{sign}(0, m)$ to obtain σ.

Bidirectional: This property is not required. (Although this property would be interesting, it does not seem likely to be achievable.)

3 Proxy Re-Signature Schemes

We begin our discussion of proxy re-signature schemes by discussing the properties of these schemes that are either necessary or simply desirable for our applications in Section 4. Next, we motivate the need for improved schemes by detailing certain inherent limitations of the original BBS [5] scheme. We then present two secure proxy re-signature schemes: *bidirectional* and *unidirectional*. The bidirectional scheme S_{bi} is based on the short signatures of Boneh et al. [10, 9]. The unidirectional scheme S_{uni} is a novel El Gamal-type algorithm over bilinear maps. We also suggest an extension to the unidirectional scheme S^*_{uni}.

3.1 Properties We Need and Want

Let us first understand what properties we expect out of a proxy re-signature scheme in addition to correctness and security from Section 2. We now informally list what are, in our opinion, the most desirable properties of a proxy re-signature scheme. In Table 1, we show which properties we are currently able to realize.

	Property	BBS [5]	S_{bi} (3.3)	S_{uni} (3.4)
1.	Unidirectional	No	No	Yes
2.	Multi-use	Yes	Yes	No
3.	Private Proxy	No	Yes	No$^\sharp$
4.	Transparent	Yes	Yes	Yes*
5.	Key Optimal	Yes	Yes	Yes
6.	Non-interactive	No	No	Yes
7.	Non-transitive	No	No	Yes
8.	Temporary	No	No	Yes*

Table 1: We compare the properties of several proxy re-signature schemes discussed in this work. The symbol * denotes that the property can be provably partially obtained or obtained by adding additional overhead. For $\sharp$, we provide some insight on how this might be achieved.

1. *Unidirectional:* The re-signature key $rk_{A\to B}$ allows the proxy to turn Alice's signatures into Bob's, but not Bob's into Alice's. This property allows for applications where the trust relationship between two parties is not necessarily mutual. Schemes that do not have this property are called *bidirectional*.

2. *Multi-use:* A message can be re-signed a polynomial number of times. That is, signatures generated by either the Sign or ReSign algorithms can be taken as input to ReSign. In constrast, one might imagine weaker, *single-use* schemes where only signatures generated by Sign can be inputs to ReSign.

3. *Private Proxy:* In a *private proxy* scheme, the re-signature keys rk's can be kept secret by an honest proxy. Thus, a single proxy may control which signatures get translated. In *public proxy* schemes, the re-signature keys can be recomputed by an adversary passively observing the proxy.

3. *Transparent:* The proxy is transparent in the scheme, meaning that a user may not even know that a proxy exists. More formally, we mean that the signatures generated by Alice on a message m using the Sign algorithm are computationally indistinguishable from her signatures on m generated by the proxy as the output of ReSign. Notice that this implies that the input and the corresponding output of the ReSign algorithm cannot be linked to each other.

5. *Key Optimal:* Alice is only required to protect and store a small constant amount of secret data (i.e., secret keys) regardless of how many signature delegations she gives or accepts. Here, we want to minimize the safe storage cost for each user. One might also wish to consider the size and number of keys that a proxy is required to safeguard.

6. *Non-interactive:* Bob (the delegator) can create the re-signature key $rk_{A\to B}$ from his secret key sk_B and Alice's *public key* pk_A, i.e., the delegatee does not act in the delegation process.

7. *Non-transitive:* The proxy alone cannot re-delegate signing rights. For example, from $rk_{A\to B}$ and $rk_{B\to C}$, he cannot produce $rk_{A\to C}$.

8. *Temporary:* Whenever a party delegates some of her rights to another party, there is always the chance that she will either need or want to revoke those rights later on. Since it may not always be feasible for a delegator to change her public key after every revocation, we are interested

in schemes that minimize revocation overhead. For example, in the case of temporary proxy re-encryption schemes, it is possible to revoke *all* delegations by changing a *single* global parameter at each time step [2]. Of course, if the re-signature proxy is trusted, then we can realize temporary delegations for any proxy re-signature scheme by issuing the appropriate instructions to the proxy.

3.2 Remarks on the BBS Scheme

In the BBS scheme, the re-signature key is necessarily public since it is not possible to store it safely at the proxy. Indeed, it would be enough to just observe a valid signature and its transformation to be able to retrieve the re-signature key and impersonate the proxy itself. Moreover, both parties are forced to mutually share their secret keys as these can be easily computed from the (public or exposed) re-signature key. Let's have a look, next, at the details of the BBS scheme so that it is easier to argue about its limitations.

BBS Re-Signatures. Recall the BBS proxy re-signature scheme [5] with global parameters (g, p, q, H), where g is a generator of a subgroup of $\mathbb{Z}_p^*$ of order $q = \Theta(2^k)$ and H is a hash function mapping strings in $\{0,1\}^*$ to elements in $\mathbb{Z}_q$.

- **Key Generation** (KeyGen)**:** On input the security parameter 1^k, select a random $a \in \mathbb{Z}_q$, and output the key pair $pk = g^a$ and $sk = a$.
- **Re-Signature Key Generation** (ReKey)**:** On input two secret keys $sk_A = a, sk_B = b$ (the public keys are not required for this algorithm), output the resignature key $rk_{A\to B} = a/b \pmod q$.
- **Sign** (Sign)**:** On input a secret key $sk = a$ and a message m, select random elements $x_1, \ldots, x_k \in \mathbb{Z}_q$. Then, compute $r = (g^{x_1}, \ldots, g^{x_k}) \pmod p$ and extract k pseudorandom bits $b_1, \ldots, b_k$ from the output of $H(r)$. Finally, output the signature $\sigma = (r, s)$, where $s = (s_1, \ldots, s_k)$ and each $s_i = (x_i - mb_i)/a \pmod q$.
- **Re-Sign** (ReSign)**:** On input a re-signature key $rk_{A\to B}$, a public key pk_A, a signature σ, and a message m, check that $\mathsf{Verify}(pk_A, m, \sigma) = 1$. If σ verifies, set $r' = r$ and $s'_i = s_i rk_{A\to B} \pmod q$, and output the signature $\sigma_B = (r', s')$, where $s' = (s'_1, \ldots, s'_k)$; otherwise, output the error message $\perp$.
- **Verify** (Verify)**:** On input a public key pk_A, a message m, and a purported signature $\sigma = (r, s)$, compute $H(r)$ and extract pseudorandom bits $b_1, \ldots, b_k$. For each $g^{x_i} \in r$ and $s_i \in s$, check that $(pk_A)^{s_i} = g^{x_i}/g^{mb_i} \pmod p$. If all check pass, output 1; otherwise output 0.

Given any pair of signatures (σ_A, σ_B), where σ_A was created by the Sign algorithm and σ_B is the result of the ReSign algorithm on σ_A, anyone can compute the re-signature key $rk_{A\to B}$ as follows: Let $\sigma_A = (r, s)$ and $\sigma_B = (r, s')$ be signatures as described above, where $s = (s_1, \ldots, s_k)$ and $s' = (s'_1, \ldots, s'_k)$. Anyone can compute $rk_{A\to B} = s'_1/s_1 = a/b \pmod q$ and become a rogue proxy. Moreover, from $rk_{A\to B}$, Alice (resp., Bob) can compute Bob's (resp., Alice's) secret key.

Although the BBS scheme satisfies their security definition (the scheme is called *symmetric* in [5]), it is clearly inadequate and cannot be used for many interesting applications, including those suggested in the original BBS paper [5].

Alternatives? Finding suitable and secure proxy re-signature schemes required a substantial effort. Natural extensions of several standard signatures were susceptible to the sort of problems above. To illustrate the intuition behind this, consider a naive proxy re-signature construction based on the popular Schnorr [23] signature. Let $||$ denote concatenation. Recall Schnorr signatures with key pairs of the form $(pk_A, sk_A) = (g^a, a)$ and signatures of the form $(r, s) = (g^k, aH(m||r) + k)$. One might think to give the proxy $rk_{A\to B} = (b - a)$, so that it can re-sign messages as $(r', s') = (r, s + rk_{A\to B}H(m||r))$. However, as in the BBS scheme, anyone can compute $rk_{A\to B} = (s' - s)/H(m||r) = (b - a)$ from a signature (r, s) and its re-signature (r, s') on a message m.

We also considered a few new schemes (e.g. [11]), but it was not obvious how to turn them into proxy re-signatures that provided a satisfying subset of the features described in Section 3.1.

3.3 S_{bi}: A Multi-Use Bidirectional Scheme

We now present a new proxy re-signature scheme, denoted S_{bi}, using the short signatures due to Boneh, Lynn, and Shacham [10, 9]. (Recall that by "short" we mean that signatures of 171 bits have roughly the same security as 1024 bit RSA signatures [10].) This scheme requires a bilinear map, as discussed in Section 2, where the map $e : G_1 \times G_1 \to G_2$ operates over two groups G_1, G_2 of prime order $q = \Theta(2^k)$. The global parameters are (e, q, G_1, G_2, g, H), where g generates G_1 and H is a hash function from arbitrary strings to elements in G_1 as defined in [7].

- **Key Generation** (KeyGen)**:** On input the security parameter 1^k, select a random $a \in \mathbb{Z}_q$, and output the key pair $pk = g^a$ and $sk = a$.
- **Re-Signature Key Generation** (ReKey)**:** On input two secret keys $sk_A = a, sk_B = b$ (the public keys are not required for this algorithm), output the resignature key $rk_{A\to B} = b/a \pmod q$.

  (Observe that the key $rk_{A\to B} = b/a$ can be securely generated in several ways, depending on the application. For example, generation of $rk_{A\to B}$ may run as follows: (1) the proxy sends a random $r \in \mathbb{Z}_q$ to Alice, (2) Alice sends r/a to Bob, (3) Bob sends rb/a to the proxy, and (4) the proxy recovers b/a. We are clearly assuming private and authenticated channels and no collusion. Bidirectional schemes make no security guarantees in the event of collusion.)
- **Sign** (Sign)**:** On input a secret key $sk = a$ and a message m, output $\sigma = H(m)^a$.
- **Re-Sign** (ReSign)**:** On input a re-signature key $rk_{A\to B}$, a public key pk_A, a signature σ, and a message m, check that $\mathsf{Verify}(pk_A, m, \sigma) = 1$. If σ does not verify, output $\perp$; otherwise, output $\sigma' = \sigma^{rk_{A\to B}}$.
- **Verify** (Verify)**:** On input a public key pk_A, a message m, and a purported signature σ, output 1 if $e(\sigma, g) = e(H(m), pk_A)$ and 0 otherwise.

Although this scheme is very simple, proving its security takes some work. We will first prove the following theorem and then discuss some of the nice properties of this scheme.

Theorem 3.1 (Security of S_{bi}) *In the random oracle model,* bidirectional *proxy re-signature scheme S_{bi} is correct and secure under the Computational Diffie-Hellman (CDH) assumption in G_1* (External and Internal Security)*; that is, for random $g \in G_1$ and $x, y \in \mathbb{Z}_q$, given (g, g^x, g^y), it is hard to compute g^{xy}.*

Proof. The correctness property is easily observable. We show security in two parts.

External Security: For security, we show that any adversary $\mathcal{A}$ that can break the security of the above proxy re-signature scheme with non-negligible probability ε after making at most q_H hash queries and requesting q_K public keys can be used to build an adversary $\mathcal{A}'$ that solves the CDH problem in G_1 with probability roughly $\varepsilon/(q_H q_K)$. On input (g, g^a, g^b), the CDH adversary $\mathcal{A}'$ simulates a proxy re-signature security game for $\mathcal{A}$ as follows:

Public keys: As $\mathcal{A}$ requests the creation of system users, $\mathcal{A}'$ guesses which one $\mathcal{A}$ will attempt a forgery against. Without loss of generality, we denote the target public key as pk_t and set it as $pk_t = g^a$, for all other public keys requested set $pk_i = g^{x_i}$ for a random $x_i \in \mathbb{Z}_q$. Let the total number of public keys requested be q_K.

Oracle Queries: There are three types of queries that $\mathcal{A}'$ must answer: the hash function H, the signature oracle $\mathcal{O}_{sign}$, and the re-signature oracle $\mathcal{O}_{resign}$.

- For each query to H on input m_i, check if there is an entry in T_H. If so, output the corresponding value, otherwise guess if m_i is the message m^* that $\mathcal{A}$ will attempt to use in a forgery. If $m_i = m^*$, output g^b; otherwise, select a random $y_i \in \mathbb{Z}_q$ and output g^{y_i}. Record the pair (m_i, y_i) in table T_H for each $m_i \neq m^*$.
- For each query to $\mathcal{O}_{sign}$ on input (j, m_i), if $j \neq t$, return the signature $H(m_i)^{x_j}$ (via calling oracle H). If $j = t$ and $m_i \neq m^*$, return the signature $(g^a)^{y_i}$; otherwise, abort.
- For each query to $\mathcal{O}_{resign}$ on input (i, j, m_k, σ), if $\mathsf{Verify}(pk_i, \sigma, m_k) \neq 1$, output $\perp$. Otherwise, output $\mathcal{O}_{sign}(j, m_k)$ (via calling oracle $\mathcal{O}_{sign}$).

Forgery: At some point $\mathcal{A}$ must output a purported forgery (j, m, σ). If $t \neq j$, then $\mathcal{A}'$ guessed the wrong target user and must abort. If $\mathsf{Verify}(pk_j, \sigma, m) \neq 1$ or (m, σ) is the result of any $\mathcal{O}_{sign}$ or $\mathcal{O}_{resign}$ query, adversary $\mathcal{A}$ has failed, so $\mathcal{A}'$ also aborts. Otherwise, $\mathcal{A}'$ outputs $\sigma = H(m^*)^a = g^{ab}$ as the proposed CDH solution.

First, we analyze how well $\mathcal{A}'$ simulates the world for $\mathcal{A}$. The responses from the hash oracle H are uniformly distributed. The responses of signing oracles $\mathcal{O}_{sign}$ and $\mathcal{O}_{resign}$ are correct, except when $\mathcal{A}'$ incorrectly guesses the target user t or the message m^* on which $\mathcal{A}$ will forge.

The probability that $\mathcal{A}'$ will guess the target user correctly is $1/q_K$. The probability that $\mathcal{A}'$ will guess the forged message m^* is $(1 - 1/2^k)/q_H$, where q_H is the number of queries made to hash function H. (The term $1 - 1/2^k$ comes from the probability that $\mathcal{A}$ will *fail* to guess $H(m^*)$ *without* querying H; obviously, $\mathcal{A}$ will fail with high probability, but in the unlikely event that $\mathcal{A}$ succeeds, the simulation will fall apart.) Recall that since the scheme is deterministic, it trivially satisfies the property that $\mathcal{A}$ cannot produce a *new* signature on a previously signed message (i.e., $\mathcal{O}_{sign}$ will *not* be forced to abort when the simulator correctly guesses the target user and message.) Thus, we conclude that if $\mathcal{A}$ forges with probability ε, then $\mathcal{A}'$ solves CDH with probability $\varepsilon(1 - 1/2^k)/(q_H q_K)$.

(Bidirectional) Internal Security: For bidirectional schemes, internal security refers only to *Limited Proxy* security; that is, a guarantee that the proxy cannot use its re-signature keys to sign on behalf of honest users. We now show that a rogue proxy $\mathcal{A}$ that can forge with probability ε can be used to build an adversary $\mathcal{A}'$ that solves the CDH problem with probability roughly $\varepsilon/(q_H q_K)$.

On input (g, g^a, g^b), the CDH adversary $\mathcal{A}'$ simulates a proxy re-signature security game for $\mathcal{A}$ as follows:

Public keys: Guess and set the target key as $pk_t = g^a$. For each key $i \neq t$, choose a random $x_i \in \mathbb{Z}_q$, and set $pk_i = (g^a)^{x_i}$. Let the total number of public keys requested be q_K.

Oracle Queries: There are three types of queries that $\mathcal{A}'$ must answer: the hash function H, the signature oracle $\mathcal{O}_{sign}$, and the re-signature key generation oracle $\mathcal{O}_{rekey}$.

- For each query to H on input m_i, check if there is an entry in T_H. If so, output the corresponding value, otherwise guess if m_i is the message m^* that $\mathcal{A}$ will attempt to use in a forgery. If $m_i = m^*$, output g^b; otherwise, select a random $y_i \in \mathbb{Z}_q$ and output g^{y_i}. Record the pair (m_i, y_i) in table T_H for each $m_i \neq m^*$.
- For each query to $\mathcal{O}_{sign}$ on input (j, m_i), if $m_i \neq m^*$, return the signature $(pk_j)^{y_i}$; else, abort.
- For each query to $\mathcal{O}_{rekey}$ on input (i, j), if $i = t$ or $j = t$, abort; else, return $rk_{i \to j} = (x_j / x_i)$.

Forgery: At some point $\mathcal{A}$ must output a purported forgery (j, m, σ). If $t \neq j$, then $\mathcal{A}'$ guessed the wrong target user and must abort. If $\mathsf{Verify}(pk_j, \sigma, m) \neq 1$ or (m, σ) is the result of any $\mathcal{O}_{sign}$ or $\mathcal{O}_{resign}$ query, adversary $\mathcal{A}$ has failed, so $\mathcal{A}'$ also aborts. Otherwise, $\mathcal{A}'$ outputs $\sigma = H(m^*)^a = g^{ab}$ as the proposed CDH solution.

The final analysis is similar to before. In the case that $\mathcal{A}'$ correctly guesses the target user t and the message to forge m^*, $\mathcal{A}'$ perfectly simulates the world for $\mathcal{A}$. Thus, CDH is solved with probability $\varepsilon(1 - 1/2^k)/(q_H q_K)$. □

Discussion of Scheme S_{bi}. This scheme is useful for many network authentication applications because it is multi-use, which allows for long signing chains. It is also bidirectional, which means that the re-signing key $rk_{A \to B}$ can be used to transform Alice's signatures into Bob's or vice versa. Bidirectionality is desirable for some applications, but a potential security risk in others. (The construction of a scheme that is both multi-use and unidirectional remains an open problem.) Transparency is guaranteed by the fact that the signature algorithm is deterministic and, since each user just stores one signing key, the scheme is also key optimal.

3.4 S_{uni} and S^*_{uni}: Single-Use Unidirectional Schemes

We now present two proxy re-signature schemes, denoted S_{uni} and S^*_{uni} respectively. These schemes are unidirectional since the re-signature key $rk_{A \to B}$ can be used to change Alice's signatures into Bob's, but *not* vice versa. The schemes S_{uni} and S^*_{uni} differ in a single feature: In S_{uni}, the re-signature key is made public or easily computable by anyone while in S^*_{uni}, this key is secret and stored at the proxy. Applications of unidirectional schemes with both public and private re-signature keys will be provided in Section 4.

Each signer has a strong and weak secret key associated to their single public key. The intuition behind the unidirectional schemes is to use the re-signature key to transform Alice's signatures computed under her strong secret into signatures computed under Bob's weak secret. Signatures under any "weak secret" cannot be converted, which makes the schemes single-use. Notice that we must deal with scenarios where signatures from several users are converted into signatures from a single user (and vice-versa). This rules out trivial solutions based on bidirectional schemes with multiple public keys per user.

3.4.1 S_{uni} with Public Re-Signature Key

Our scheme requires a bilinear map, as discussed in Section 2, where the map $e : G_1 \times G_1 \rightarrow G_2$ operates over two groups G_1, G_2 of prime order $q = \Theta(2^k)$. The global parameters are $(e, q, G_1, G_2, g, h, H)$, where g and h are generators of G_1 and H is a hash function from arbitrary strings to elements in $\mathbb{Z}_q$.

- **Key Generation** (KeyGen)**:** On input the security parameter 1^k, select a random $a \in \mathbb{Z}_q$, and output the key pair $pk = (g^a, h^{1/a})$ and $sk = a$. We think of $sk = a$ as the "strong" secret, and the value h^a as the "weak" secret.

  (Note: a user need only output the second component $h^{1/a}$ of her public key if she wishes to receive delegations; it does not play a role in signature verification. Also, the second component can be verified against the first as $e(g^a, h^{1/a}) = e(g, h)$.)

- **Re-Signature Key Generation** (ReKey)**:** On input a public key $pk_A = (g^a, h^{1/a})$ and a secret key $sk_B = b$, output the re-signature key $rk_{A \rightarrow B} = h^{b/a}$. Let $rk_{A \rightarrow B}$ be public for this scheme.

- **Sign** (Sign)**:** On input a secret key $sk = a$ and a message m, select a random $k \in \mathbb{Z}_q$, set $r = h^k$, $s = a(H(m||r) + k) \pmod q$; output the pair $\sigma = (r, s)$. We call a signature of this form a *first-level* signature.

  Optionally, the signer could choose to output a signature that could not be re-signed, where the last element of σ is set to $h^{aH(m||r)+ak}$ instead. We call this a *second-level* signature.

- **Re-Sign** (ReSign)**:** On input a re-signature key $rk_{A \rightarrow B}$, a public key pk_A, a (first-level) signature σ, and a message m, check that $\mathsf{Verify}(pk_A, m, \sigma) = 1$. If $\sigma = (r, s)$ does not verify, output $\perp$; otherwise, set $r' = r$ and $s' = (rk_{A \rightarrow B})^s$, and output $\sigma' = (r', s')$.

- **Verify** (Verify)**:** On input a public key $pk = (pk^{(1)}, pk^{(2)})$, a message m, and a purported signature $\sigma = (r, s)$ (if σ is a first-level signature, set $s = h^s$), output 1 if $e(g, s) = e(pk^{(1)}, rh^{H(m||r)})$ and 0 otherwise.

Theorem 3.2 (Security of S_{uni}) *In the random oracle model,* unidirectional *proxy re-signature scheme S_{uni} is correct and secure under the CDH and 2-DL assumptions in G_1* (External and Internal Security)*; the latter that given (g, g^a, g^{a^2}), for random $g \in G_1$ and $a \in \mathbb{Z}_q$, it is hard to compute a.*

Proof. The correctness property is easily observable. We argue security in two parts. Here, for clarity of exposition, we will use the well-known result that CDH is equivalent to the *Square Diffie-Hellman* (sq-DH) problem in the same group. The sq-DH problem in G_1 is to compute g^{x^2} given (g, g^x) for a random $g \in G_1$ and $x \in \mathbb{Z}_q$. Showing that sq-DH implies CDH is trivial. To see the other direction, suppose you are given CDH input (g, g^x, g^y) and an sq-DH solver. Use the sq-DH solver to compute $A = g^{x^2}$, $B = g^{y^2}$, and $C = g^{(x+y)^2} = g^{x^2+2xy+y^2}$. From these values, it is easy to compute $\sqrt{C/(AB)} = g^{xy}$ for this group of prime order.

External Security: In S_{uni}, the re-signature keys are public by design. Thus, all users are considered proxies and there are no "external adversaries" to the system. Therefore, the external security property holds trivially.

(Unidirectional) Internal Security: For unidirectional schemes, internal security refers to:

- *Limited Proxy Security* protecting an honest delegator and delegatee from a rogue proxy,
- *Delegatee Security* protecting an honest delegatee against delegators colluding with the proxy,
- *Delegator Security* offering more limited protection to an honest delegator against delegatees colluding with the proxy.

Limited Proxy Security: For security, we show that any proxy $\mathcal{A}$ that can break this security guarantee with non-negligible probability ε can be used to build an adversary $\mathcal{A}'$ that solves the CDH (equivalently, sq-DH) problem in G_1 with probability roughly ε^2.

System Parameters: On a sq-DH challenge (g, g^x), the simulator $\mathcal{A}'$ outputs the system parameters as g and $h = g^x$.

Public keys: As $\mathcal{A}$ requests the creation of system user i, $\mathcal{A}'$ chooses a random $y_i \in \mathbb{Z}_q$ and outputs $pk_i = (g^{xy_i}, g^{1/y_i}) = (g^{xy_i}, h^{1/(xy_i)})$. The virtual sk_i is xy_i. The pair (i, y_i) is saved.

Oracle Queries: There are three types of queries that $\mathcal{A}'$ must answer: the hash function H, the signature oracle $\mathcal{O}_{sign}$, and the re-signature key generation oracle $\mathcal{O}_{rekey}$.

- For each query to H on input a_i, check if there is an entry in T_H. If so, output the corresponding value, otherwise output a random value $c_i \in \mathbb{Z}_q$. Record the pair (a_i, c_i) in table T_H. Let q_H be the total number of queries to H.
- For each query to $\mathcal{O}_{sign}$ on input (j, m_i), concoct the signature using control over the output of H as follows. Select random values $s, c \in \mathbb{Z}_q$. Parse user j's public key as $pk_j = (pk_j^{(1)}, pk_j^{(2)})$. Compute $r = (pk_j^{(2)})^s / h^c$. Check if $(r||m_i, \cdot)$ is already in T_H, if so abort; otherwise, output the first-level signature (r, s). Record $(r||m_i, c)$ in table T_H.
- For each query to $\mathcal{O}_{rekey}$ on input (i, j), generate the re-signature key $rk_{i \to j}$ by computing $(g^x)^{y_j/y_i}$. (This is equal to $h^{(xy_j)/(xy_i)} = h^{sk_j/sk_i}$.)

In the above, $\mathcal{A}'$ *almost* perfectly simulates the world for adversary $\mathcal{A}$, except for the possibility of aborting in oracle H. Aborting happens on collisions in H which occur with probability at most $q_H/2^k$; this probability could be further reduced by choosing new values for s and c in the event of a collision and trying again. Thus a simulation will end with $\mathcal{A}$ successfully forging some message with probability $\varepsilon(1 - q_H/2^k)$. Applying the Reset Lemma [4], we know that with probability at least $(\varepsilon - (\varepsilon q_H + 1)/2^k)^2$, $\mathcal{A}$ can produce two valid signature transcripts (r, c_1, s_1) and (r, c_2, s_2) for user t, and where c_1 and c_2 are two different random responses from H on input $(r||m)$ for some m. When this occurs, $\mathcal{A}$ can solve sq-DH by computing and outputting:

$$\left(\frac{s_1}{s_2}\right)^{1/(y_i(c_1-c_2))} = \left(\frac{h^{xy_i(c_1+k)}}{h^{xy_i(c_2+k)}}\right)^{1/(y_i(c_1-c_2))} = h^{\frac{xy_i(c_1+k-c_2-k)}{y_i(c_1-c_2)}} = h^x = (g^x)^x = g^{x^2}.$$

Delegatee Security: Recall that S_{uni} is a non-interactive scheme, meaning that Bob (the delegator) can compute a re-signature key $rk_{A \to B}$ from Alice's public key. Thus, intuitively "nothing is learned about Alice's secrets" from Bob and the proxy both seeing $rk_{A \to B}$.

More formally, $\mathcal{A}'$ must be able to provide $\mathcal{A}$ with the secret keys of all delegators of a target user pk_0. Here $\mathcal{A}$ need only produce a second-level signature to "win" in the delegatee security game.

System Parameters: On a sq-DH challenge (g, g^a), the simulator $\mathcal{A}'$ outputs the system parameters as g and $h = (g^a)^x$ for random $x \in \mathbb{Z}_q$.

Public and Secret Keys: $\mathcal{A}'$ generates the following keys for $\mathcal{A}$.

- For the delegatee, set the public key as $pk_0 = (g^a, h^{1/a} = g^x)$. For all other users, set as $pk_i = (g^{y_i}, h^{1/y_i} = g^{x/y_i})$ for a random $y_i \in \mathbb{Z}_q$.
- Output each delegator's secret key sk_i as y_i.

Oracle Queries: There are three types of queries that $\mathcal{A}'$ must answer: the hash function H, the signature oracle $\mathcal{O}_{sign}$, and the re-signature key oracle $\mathcal{O}_{rekey}$.

- For each query to H on input x_i, check if there is an entry in T_H. If so, output the corresponding value, otherwise output a random value $c_i \in \mathbb{Z}_q$. Record the pair (x_i, c_i) in table T_H.
- For each query to $\mathcal{O}_{sign}$ on input (j, m_i), select random values $s, c \in \mathbb{Z}_q$. Parse user j's public key as $pk_j = (pk_j^{(1)}, pk_j^{(2)})$. Compute $r = (pk_j^{(2)})^s / h^c$. Check if $(r||m_i, \cdot)$ is already in T_H, if so abort; otherwise, Output the first-level signature (r, s). Record $(r||m_i, c)$ in table T_H.
- For each query to $\mathcal{O}_{rekey}$ on input $(0, i)$, compute each re-signature key $rk_{0\to i}$ as $(h^{1/a})^{y_i}$. All other allowed keys $rk_{i\to j}$ (i.e., $j \neq 0$) $\mathcal{A}$ can compute himself given the secret keys.

Adversary $\mathcal{A}'$ simulation of the world for $\mathcal{A}$ succeeds with the same probability as before, thus we apply the Reset Lemma [4], and from the two second-level signature transcripts compute $h^a = g^{a^2}$. (Note that $\mathcal{A}'$ only aborts during $\mathcal{O}_{sign}$ in the unlikely event that a collision occurs; unlike the proof of Theorem 3.1, it does not depend on *which* message is being signed. Thus, $\mathcal{A}'$ succeeds even when $\mathcal{A}$ produces a new forgery for a message $\mathcal{A}'$ was already asked to sign.)

Delegator Security: $\mathcal{A}'$ must now be able to provide $\mathcal{A}$ with the secret keys of all delegatees of a target user pk_0. Since $\mathcal{A}$ must produce a first-level signature to "win" in the delegator security game, we actually base this property on the 2-DL assumption that given (g, g^a, g^{a^2}), for random $g \in G_1$ and $a \in \mathbb{Z}_q$, it is hard to compute a.

System Parameters: On a 2-DL challenge (g, g^a, g^{a^2}), the simulator $\mathcal{A}'$ outputs the system parameters as g and $h = (g^a)^x$ for random $x \in \mathbb{Z}_q$.

Public and Secret Keys: $\mathcal{A}'$ generates the following keys for $\mathcal{A}$.

- For the delegator, set the public key as $pk_0 = (g^a, g^x)$. For all other users, set as $pk_i = (g^{y_i}, h^{1/y_i} = g^{x/y_i})$ for a random $y_i \in \mathbb{Z}_q$.
- Output each delegatee's secret key sk_i as y_i.

Oracle Queries: There are three types of queries that $\mathcal{A}'$ must answer: the hash function H, the signature oracle $\mathcal{O}_{sign}$, and the re-signature key oracle $\mathcal{O}_{rekey}$.

- For each query to H on input x_i, check if there is an entry in T_H. If so, output the corresponding value, otherwise output a random value $c_i \in \mathbb{Z}_q$. Record the pair (x_i, c_i) in table T_H.

- For each query to $\mathcal{O}_{sign}$ on input (j, m_i), select random values $s, c \in \mathbb{Z}_q$. Parse user j's public key as $pk_j = (pk_j^{(1)}, pk_j^{(2)})$. Compute $r = (pk_j^{(2)})^s / h^c$. Check if $(r||m_i, \cdot)$ is already in T_H, if so abort; otherwise, Output the first-level signature (r, s). Record $(r||m_i, c)$ in table T_H.
- For each query to $\mathcal{O}_{rekey}$ on input $(i, 0)$, compute each re-signature key $rk_{i\to 0}$ as $(g^{a^2})^{x/y_i} = h^{a/y_i}$. All other keys $rk_{i\to j}$ the adversary $\mathcal{A}$ can compute himself given the secret keys.

The simulation succeeds with the same probability as before, thus we apply the Reset Lemma [4], and from the two first-level signature transcripts compute $sk_0 = a$.

□

3.4.2 S^*_{uni} with Private Re-Signature Key

In S_{uni}, re-signature keys are *public*. This does not render the system as vulnerable as the BBS scheme, since at least the delegatee remains secure. For many of the applications we will shortly discuss in Section 4, our schemes S_{bi} and S_{uni} will be sufficient. However, it would also be desirable to have a unidirectional scheme where the proxy key can be kept private. We briefly propose how one might consider naturally modifying S_{uni} into a new scheme S^*_{uni} to achieve these properties. The setup and global parameters hold from S_{uni}. The following algorithms change:

- **Re-Signature Key Generation** (ReKey^*)**:** The re-signature key is $rk_{A\to B} = h^{b/a}$ as before, plus the proxy stores pk_B. The proxy keeps $rk_{A\to B}$ private.
- **Re-Sign** (ReSign^*)**:** On input a re-sig. key $rk_{A\to B}$, a public key pk_A, a (first-level) signature σ, and a message m, check that $\mathsf{Verify}(pk_A, m, \sigma) = 1$. If $\sigma = (r, s)$ does not verify, output $\perp$; otherwise choose a random $w \in \mathbb{Z}_q$, set $r' = r$, $s' = (rk_{A\to B})^{sw}$, $t' = (pk_B^{(1)})^w$, and generate a signature proof of knowledge u' of the discrete logarithm of t' for base $pk_B^{(1)}$ (i.e., the first part of Bob's public key) on message (r', s', t') using a new global hash function $\hat{H} : \{0,1\}^* \to \mathbb{Z}_q$. (This last step can be efficiently done using Schnorr's technique [23].) Output $\sigma' = (r', s', t', u')$.
- **Verify** (Verify^*)**:** The verifier now checks: (1) the proof u', and (2) $e(g, s') = e(t', r' h^{H(m||r')})$.

The scheme S^*_{uni} is a natural extension of S_{uni} and we conjecture its security is based on the same assumptions. We leave a formal analysis of it as a subject for future work.

Discussion of Schemes S_{uni} and S^*_{uni}. The only conceptional difference between these two schemes is that in S_{uni} the re-signature key is necessarily public (i.e., it is prey to the Section 3.2 attack), while in S^*_{uni} the proxy can keep the re-signature key private. (The re-randomization added to ReSign^* thwarts this attack.)

Even though the re-signature key $rk_{A\to B}$ in S_{uni} is public, which allows anyone to translate between signatures, it does not reveal any information about Alice's (delegatee) signing keys to *anyone* (because it was computed by Bob with her public key). This is important since the delegatee should not be taking on any security risk. Furthermore, no third party can use this key to sign arbitrary messages for Bob (delegator) – and Alice can only recover Bob's *weak* secret h^b. This does not give Alice any new signing capability that she didn't have before: Alice could sign on behalf of Bob anyway, either by herself (S_{uni}) or jointly with the proxy (S^*_{uni}). (We stress that Alice won't be able to generate Bob's first-level signatures in any case.)

Bob does run the risk, however, that Alice may publish h^b, allowing anyone to produce second-level signatures for Bob. Yet, such a risk for the delegator seems inevitable.

Both of these schemes are exclusively for single-use applications (i.e., a signature translated from Alice to Bob cannot be again translated from Bob to Carol). One such application is a company mail server turning employee's signatures into a group signature before public release. An interesting open problem is designing a scheme that is simultaneously unidirectional and multi-use.

Happily, these schemes are non-interactive since Bob only needs Alice's public key to delegate to her (i.e., $rk_{A \to B}$). One potential drawback is that the original and re-signed values can be linked; that is, given a first-level signature pair (r, s), the ReSign algorithm produces a new second-level signature pair (r', s') (or (r', s', t', u')) with $r = r'$. Nevertheless, weak transparency is achieved because the delegator can also produce second-level signatures from first-level ones due to the fact that he knows the re-signature key.

Temporary Delegations (and Revocations). What if Bob only wishes to allow Alice's signatures to be turned into his, via giving $rk_{A \to B}$ to the proxy, for the span of one week? If the proxy is honest, Bob may simply issue it these instructions. However, if Bob does not fully trust the proxy, is his only option to change his public key? Fortunately, the answer is no. Applying techniques similar to those by Ateniese et al. [2] for building temporary re-encryption schemes, we propose the following change to our schemes. At each time period i, a trusted party broadcasts a new global parameter $h_i \in G_1$, which replaces $h_{(i-1)}$ in the signing, verification, and re-signature key generation algorithms. This method effectively revokes *all* re-signature keys at each time period, but takes only a single broadcast value and leaves all *certified* public keys valid.

4 Applications

Blaze, Bleumer, and Strauss [5] suggested several interesting applications of proxy re-signatures relating to key management. We begin by taking a closer look at how proxy re-signatures can help relieve some of the common key management headaches. Next, we explore a broader set of applications and show that proxy re-signatures can be used to form weak (but easy to manage) group signatures, a space-efficient proof that a certain path was taken in a graph (e.g., a packet followed a prescribed path through the network), and more.

4.1 Exploring BBS Key Management

In particular, BBS [5] pointed out that proxy re-signatures can be used to change or add new public key pairs to a system without obtaining new certificates, notably simplifying key management. Let us explore this idea in more detail.

Certifying Keys is Expensive, Can We Share? Since certification of new public keys is a procedure that can be expensive and time consuming, using proxy re-signatures is a way to *share* existing certificates. Signatures under new keys can be transformed into ones that can be verified with public keys that are already certified. Consider also that distribution of certificates may be difficult or impossible in certain environments. Proxy re-signatures could be used to mitigate (at least temporarily) this issue by transforming signatures into ones that can be verified with public keys already trusted by the verifier. We now present an example of certificate sharing.

A Time to Share (using S_{uni} or S^*_{uni}). Consider the case where a set of public keys is embedded into a software product or a limited-storage device, such as a smartcard. In many cases, it would be convenient if operating systems came installed with the certified public keys of major companies. The drawback, however, is that it might then be difficult or cumbersome for a company to change or add new keys. For example, a large company may want to begin signing documents at a department, rather than a company-wide, level even though software has been shipped with only one company-wide verification key pk_A. To solve this dilema, the company could set up a proxy which could translate between old and new keys or from a large set of keys to a smaller one, etc.

To see this, suppose that software was shipped with a single company verification key pk_A. The company could still create new verification keys for each department pk_B, pk_C, pk_D and include these cerified keys in the next software release. However, to temporarily remain backwards compatible with the old software, the company could also publish (or setup a semi-trusted proxy) with the re-signature keys $rk_{B\to A}$, $rk_{C\to A}$, $rk_{D\to A}$; thus the proxy could change any signature generated by departments B, C, or D (which the old software would not recognize) and turn it into a company-wide signature under A (which the old software will recognize).

Where Previous Schemes Fail in These Applications. Although (some form of) the applications presented above were proposed for the BBS proxy re-signature scheme [5], that scheme is not suitable for them given that in the process of certificate-sharing both the original, specialized signature $\sigma_B(m)$ (for the updated software) and the re-signed, company-wide signature $\sigma_A(m)$ (for the old software) may be public. As we saw in Section 3.2, *anyone* can compute the proxy's "secret" re-signing key $rk_{A\to B}$ after seeing a signature $\sigma_A(m)$ and its translation $\sigma_B(m)$. Moreover, an active adversary will have an easy time finding an original and re-signed pair of signatures using the BBS scheme, since the re-signatures are *linkable* to their originals; that is, the signature (r, s), given as input to the ReSign algorithm, and its corresponding output (r', s') have $r' = r$ set by the protocol specification.

4.2 New Applications

Armed with our new proxy re-signature schemes, we now show that they can be used as a space-efficient "proof" that a path was taken in a graph, or as an easy-to-manage group signature scheme, and to simplify certificate management.

Space-Efficient Proof that a Path was Taken (using S_{bi}). Proxy re-signatures are particularly useful when deployed with their multi-use capability, such as S_{bi}. In particular, signatures can be converted in series as shown in Figure 1. Here, the signer A generates the first signature on the message m, $\sigma_A(m)$, and the intermediate proxies convert it into the final signature $\sigma_E(m)$ through a series of transformations repeated in sequence. Such a structure can be used to prove that a certain item followed a specific path without taking any shortcuts.

The United States is currently in the process of adopting *E-passports* [17] – traditional passport documents capable of storing a limited number of digital signatures. Suppose Eve arrives in New York from her home country of Eden and shows US border patrol a signature $\sigma_A(m)$ from Eden that she is a citizen in good standing. The border patrol officer checks this signature and translates it into $\sigma_B(m)$, stating that Eve has passed the border patrol check. Eve next takes her passport to the customs officer. The customs officer need only verify Eve's passport against one public key – that of border patrol – and if it checks out and she passes customs, he can translate the signature into $\sigma_C(m)$, etc.

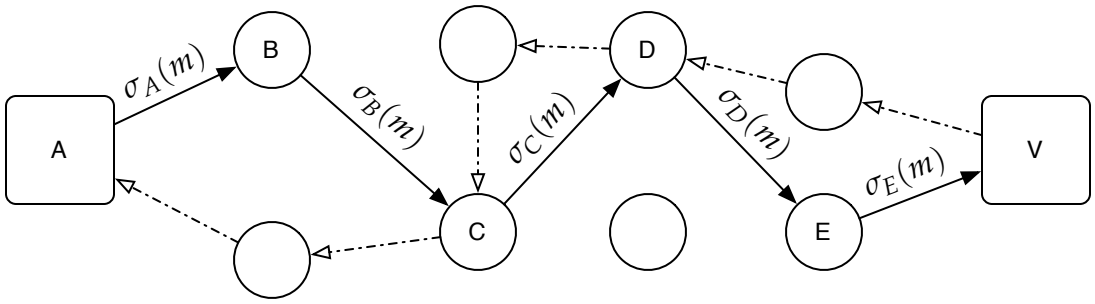

Figure 1: Unidirectional chains with multi-use proxy re-signature. Each node (e.g., C) holds only a proxy re-signature key (e.g., $rk_{C\to D}$) and *not* a signing key (e.g., sk_C). Thus, an adversary cannot inject a signed message into the chain by corrupting intermediate nodes.

This system has many benefits. First, keeping only one signature around at a time reduces the space requirements on the limited memory passport – and also reduces the number of verification keys that checkpoints down the chain must store. Second, by only giving each checkpoint a re-signature key (e.g., giving customs $rk_{B\to C}$ instead of sk_C), corrupting a customs officer only allows Eve to skip the customs check – but she must still have gone through the intial checks by Eden and border patrol. Thus, Eve can – at best – skip one stop for each checkpoint officer that she compromises, and only for a message that Eden already authenticated.

Notice that although we use our bidirectional scheme (because it is "multi-use"), the chain in Figure 1 is actually *unidirectional* as long as the secret e corresponding to the last public key is kept secret. Thus, we can design only one way for Eve to get through the airport checks. Obviously, a scheme that is both multi-use and unidirectional would be ideal for this application, but no such scheme currently exists. Fortunately, by the strategic release of certain S_{bi} re-signature keys we can design for arbitrary traversal of a given "graph" allowing for a host of other *quality check* applications.

With a proper release of keys, proxy re-signatures can be used to non-repudiably prove that a message has traversed a graph via a legitimate path. Of course, one could employ *multi-signatures* [19] or *aggregate signatures* [8] to get a proof that certain nodes were visited, but both of these solutions require the verifier to have the verification keys of *all* of these nodes. Using proxy re-signatures, each node (including the final one) need only store and trust a *single* public key (the public key of the node preceeding it in the chain) – and yet each node has some real confidence that it is validating the *entire* path. Thus, we see another savings in key management.

Additionally in some cases, users may want the *privacy* that multi-use proxy re-signatures provide; that is, these signatures could simultaneously *authenticate and yet hide* the path traversed by the message in the network.

Easy to Manage Group Signatures (using S^*_{uni}). Proxy re-signatures can be used to conceal identities or details of the structure of an organization. For instance, a corporate proxy sitting on a company's outgoing mail server could translate the individual signatures of its employees, which are perfectly valid signatures inside the organization, into signatures that can be verified with a single *corporate* public key. The proxy could (optionally) log which employee signed the message for internal auditing, but choose to keep that information company confidential. The interesting

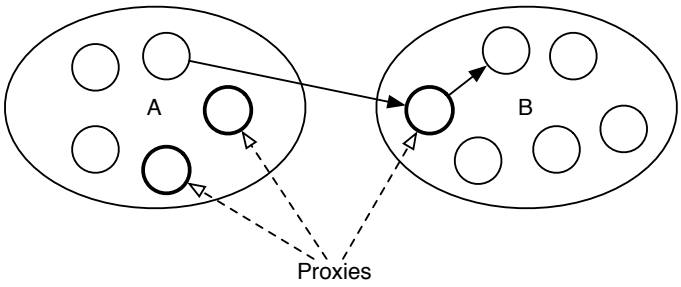

Figure 2: Using a unidirectional re-signature scheme, two ad-hoc networks can set up their own trusted "proxies", inside their own perimeter or domain, to translate incompatible certificates issued by an authority for the other network.

feature here is that even if the proxy is compromised *from the outside*, no signing keys are ever revealed which makes the proxy a less appealing target. The actual corporate secret key could be kept securely in a lock-box, and then the proxy, with only re-signing information, could sit out on the mail server. For accountability purposes, it is also advisable to employ unidirectional schemes (with private re-signature key) so that the proxy will not be able to generate members' signatures from existing corporate signatures.

Transparent Certification (using S_{bi} or, in some cases either, S_{uni} or S^*_{uni}). Proxy re-signatures can be used to translate between public-key certificates, which are often implemented as digital signatures, from different Certification Authorities (CAs). Suppose A and B want to communicate securely by building a private and authenticated channel. The common first step for A and B would be to exchange their certified public keys. However, let's assume that A can only verify certificates from the certification authority CA1 and B those from CA2.

A proxy could be set up by the certification authorities to temporarily convert certificates from CA1 to ones from CA2 and vice-versa. This approach is useful when a high level of coordination between A and B is impractical, or when two distinct entities suddenly agree to cooperate, e.g., two banks merge into a single company and deploy (or publish) proxy keys as a temporary compatibility fix.

We notice that there are certain advantages in using unidirectional schemes, rather than bidirectional ones, when setting up such a proxy. For example, consider a scenario in which A and B are two (ad-hoc) networks that use their own CA and have their own domain, as shown in Figure 2. We can achieve the same functionality of a bidirectional scheme by using a unidirectional one with two re-signature keys, $rk_{A\to B}$ and $rk_{B\to A}$.

For example, as shown in Figure 2, each time a certificate is sent from one network to another, it is first processed and translated by a proxy in the destination network and forwarded directly to the destination node. Alternatively, the destination node could collect the incompatible certificate from the source node of the other network and forward it to one of its *local* proxies to have it translated.

The main advantage of using proxy re-signature schemes in this scenario is to allow the two networks to communicate in a way that is completely transparent to the internal nodes. Indeed, nodes do not have to be reprogrammed or instructed to trust and store new keys (certain types of sensor or RFID chips cannot be reprogrammed or do not have enough memory to store new information). The advantage of using unidirectional schemes over bidirectional ones is to allow

each network to set up proxies inside their own perimeter or domain without relying on nodes from the other (possibly untrusted) network. Moreover, our approach does not require interaction since each network can generate a re-signature key directly from the public key of the other network.

5 Conclusions

In this work, we formalized the proxy re-signature primitive of Blaze, Bleumer, and Strauss [5]. We pointed out several limitations of the BBS scheme and we provided new improved constructions. One of our schemes (S_{uni}) allows the proxy to translate from Alice to Bob, but not vice versa. This is the only known construction to have this property since BBS proposed the concept in 1998 [5]. Our schemes are efficient and based on standard assumptions in the random oracle model, although they offer slightly different properties. Finally, we presented exciting applications of proxy re-signatures, including key management, (weak) group signatures, and short proofs that a valid path was taken in a graph. We are confident that proxy re-signatures have many additional applications beyond those mentioned here.

One open problem of particular interest stemming from this work is whether or not proxy re-signature schemes can be built that translate from one type of signature scheme to another. For example, a scheme that translates Alice's Schnorr signatures into Bob's RSA-based ones. The existence of multi-use, unidirectional schemes also remains open.

Acknowledgments. We are grateful to Kevin Fu, Matthew Green, Ari Juels, and Adam Stubblefield for discussions on applications of proxy re-signatures. In particular, Kevin Fu suggested to use proxy re-signatures to non-repudiably prove that a *maze* was solved. We thank Dan Boneh for useful comments on assumptions used in earlier schemes. A preliminary version of this paper was accepted at ACM CCS 2005, we thank the CCS anonymous reviewers for their insightful comments.

Susan Hohenberger's work was supported by an NDSEG Fellowship.

References

[1] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In *Advances in Cryptology – EUROCRYPT '02*, volume 2332 of LNCS, pages 83–107, 2002.

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

[3] Mihir Bellare and Gregory Neven. Transitive Signatures Based on Factoring and RSA. In *Advances in Cryptology – ASIACRYPT '02*, volume 2501 of LNCS, pages 397–414, 2002.

[4] Mihir Bellare and Adriana Palacio. GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, volume 2442 of LNCS, pages 162–177. Springer Verlag, 2002.

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

[6] A. Boldyreva. Efficient Threshold Signature, Multisignature, and Blind Signature Schemes based on the Gap-Diffie-Hellman-group signature Scheme. In *Public Key Cryptography 2003*, volume 2567 of LNCS, pages 31–46, 2003.

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

[8] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures. In *Advances in Cryptology – EUROCRYPT '03*, volume 2656 of LNCS, pages 416–432, 2003.

[9] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil Pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

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

[11] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology — CRYPTO 2004*, volume 3152 of LNCS, pages 56–72, 2004.

[12] Yevgeniy Dodis and Anca Ivan. Proxy cryptography revisited. In *Network and Distributed System Security Symposium*, February 2003.

[13] Yevgeniy Dodis and Leonid Reyzin. Breaking and repairing optimistic fair exchange from PODC 2003. In *Proceedings of the Third ACM Workshop on Digital Rights Management (DRM'03)*, ACM Press, 2003.

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

[15] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, 1988.

[16] Antoine Joux. A one-round protocol for tripartite Diffie-Hellman. In *ANTS-IV conference*, volume 1838 of LNCS, pages 385–394, 2000.

[17] Ari Juels, David Molnar, and David Wagner. Security and privacy issues in e-passports. In *IEEE SecureComm '05 (to appear)*, 2005.

[18] M. Mambo, K. Usuda, and E. Okamoto. Proxy signatures: delegation of the power to sign messages. *IEICE Trans. Fundamentals*, E79-A(9), 1996.

[19] S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures. In *ACM Communication and Computer Security 2001*, pages 245–54. ACM press, 2001.

[20] Silvio Micali and Ronald L. Rivest. Transitive signature schemes. In *CT-RSA '02*, volume 2271 of LNCS, pages 236–243, 2002.

[21] K. Ohta and T. Okamoto. Multisignature schemes secure against active insider attacks. *IEICE Trans. Fundamentals*, E82-A/1:21–31, 1999.

[22] T. Okamoto. A digital multisignature scheme using bijective public-key cryptosystems. *ACM Trans. Computer Systems*, 6(4):432–441, 1998.

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