Atomic Proxy Cryptography

Matt Blaze Martin Strauss
AT&T Labs – Research
Florham Park, NJ 07932
{mab,mstrauss}@research.att.com

DRAFT – 2 November 1997 – DO NOT RE-DISTRIBUTE*

Abstract

This paper introduces *atomic proxy cryptography,* in which an *atomic proxy function,* in conjunction with a public *proxy key,* converts ciphertext (messages in a public key encryption scheme or signatures in a digital signature scheme) for one key (k_1) into ciphertext for another (k_2). Proxy keys, once generated, may be made public and proxy functions applied in untrusted environments. Various kinds of proxy functions might exist; *symmetric* atomic proxy functions assume that the holder of k_2 unconditionally trusts the holder of k_1, while *asymmetric* proxy functions do not. It is not clear whether proxy functions exist for previous public-key cryptosystems. Several new public-key cryptosystems with symmetric proxy functions are described: an encryption scheme, which is at least as secure as Diffie-Hellman, an identification scheme, which is at least as secure as the discrete log, and a signature scheme derived from the identification scheme via a hash function.

1 Introduction

1.1 Overview

A basic goal of public-key encryption is to allow only the key or keys selected at the time of encryption to decrypt the ciphertext. To change the ciphertext to a different key requires re-encryption of the message with the new key, which implies access to the original cleartext and to a reliable copy of the new encryption key. Intuitively, this seems a fundamental, and quite desirable, property of good cryptography; it should not be possible for an untrusted party to change the key with which a message can be decrypted.

This paper, on the other hand, investigates the possibility of *atomic proxy functions* that convert ciphertext for one key into ciphertext for another without revealing secret decryption keys or cleartext messages. An atomic proxy function allows an untrusted party to convert ciphertext between keys without access to either the original message or to the secret component of the old key or the new key. In proxy cryptography, the holders of public-key pairs A and B create and publish a *proxy key* $\pi_{A\to B}$ such that $D(\Pi(E(m,e_A),\pi_{A\to B}),d_B)=m$, where $E(m,e)$ is the public encryption function of message m under encryption key e, $D(c,d)$ is the decryption function of ciphertext c under decryption key d, $\Pi(c,\pi)$ is the atomic proxy function that converts ciphertext

*Current draft available at ftp://ftp.research.att.com/dist/mab/proxy.ps

c according to proxy key π, and e_A, e_B, d_A, d_B are the public encryption and secret decryption component keys for key pairs A and B, respectively. The proxy key gives the owner of B the ability to decrypt "on behalf of" A; B can act as A's "proxy." In other words, the Π function effectively allows the "atomic" computation of $E(D(c, d_A), e_B)$ without revealing the intermediate result $D(c, d_A)$.

We consider atomic proxy schemes for encryption, identification and signatures. An encryption proxy key $\pi_{A \to B}$ allows B to decrypt messages encrypted for A and an identification or signature proxy key $\pi_{A \to B}$ allows A to identify herself as B or to sign for B (i.e., transforms A's signature into B's signature). Generating encryption proxy key $\pi_{A \to B}$ obviously requires knowledge of at least the secret component of A (otherwise the underlying system is not secure) and similarly generating identification or signature proxy key $\pi_{A \to B}$ requires B's secret, but the proxy key itself, once generated, can be published safely.

1.2 Categories of proxy schemes

Encryption proxy functions (and similarly but contravariantly, identification or signature proxy functions) can be categorized according to the degree of trust they imply between the two key holders. Clearly, A must (unconditionally) trust B, since the encryption proxy function by definition allows B to decrypt on behalf of A. *Symmetric* proxy functions also imply that B trusts A, e.g., because d_B can be feasibly calculated given the proxy key plus d_A. *Asymmetric* proxy functions do not imply this bilateral trust. (Note that this model implies that proxy cryptography probably makes sense only in the context of public-key cryptosystems. Any secret-key cryptosystem with an asymmetric proxy function could be converted into a public-key system by publishing one key along with a proxy key that converts ciphertext for that key into ciphertext for a second key (which is kept secret.))

We can also categorize the asymmetric proxy schemes that might exist according to the convenience in creating the proxy key. In an *active asymmetric* scheme, B has to cooperate to produce the proxy key $\pi_{A \to B}$ feasibly, although the proxy key (even together with A's secret key) might not compromise B's secret key. In a *passive asymmetric* scheme, on the other hand, A's secret key and B's public key suffice to construct the proxy key. Clearly, any passive asymmetric scheme can be used as an active asymmetric scheme, and any asymmetric scheme can be used as a symmetric scheme.

Finally, we can distinguish proxy schemes according to the "metadata" they reveal about the identity of the keys being transformed. *Transparent* proxy keys reveal the original two keys to a third party. *Translucent* proxy keys allow a third party to verify a guess as to which two keys are involved (given their public keys). *Opaque* proxy keys reveal nothing, even to an adversary who correctly guesses the original keys (but who does not know the private keys involved).

1.3 Proxy schemes in theory and practice

The proxy relationship is necessarily transitive. If there are public proxy keys $\pi_{A \to B}$ and $\pi_{B \to C}$, then anyone can compute a proxy function for $A \to C$. Symmetric proxy schemes further establish equivalence classes of keys where the secret component of any key can be used to decrypt messages for any other key in the same class. Note that creating a single symmetric proxy key between a key in one class and a key in another effectively joins the two classes into one.

The notion of proxy cryptography is a rather natural generalization of public-key cryptography and has some pleasing theoretical properties. The proxy schemes we consider below have the additional property that anyone can use the proxy key $\pi_{A\to B}$ to transform the public key of A to the public key of B. For such proxy schemes, as we will see in the various examples below, certain aspects of the security of publishing a proxy key actually follow from the fact that anyone, trusted or not, can use a proxy key to transform ciphertext and keys.

For example, suppose random messages m and m' are encrypted with random secret keys a and b as $E(m, a)$, $E(m', b)$. Suppose that knowing the proxy key $\pi_{A\to B}$ enables Eve, who knows neither a nor b, to recover m or m'. Then, ignoring B altogether and starting with just two (presumably secure) ciphertexts $E(m, a)$ and $E(m', a)$, Eve can pick a random proxy key $r = \pi_{A\to Q}$ for some Q, transform $E(m', a)$ to $E(m', q)$ (where q is the unknown secret key of Q), transform A's public key into Q's public key, and proceed with the hypothesized cryptanalysis. We conclude that if it is safe for A to publish k messages then it is safe for A and B to publish a total of k messages *and* to publish a proxy key, provided only that Eve can successfully *apply* the proxy key to transform ciphertext and public keys.

Because proxy keys are tied to specific key pairs, it is not necessary in many applications to certify or otherwise take special care in distributing them (except to prevent denial-of-service). In particular, it is generally sufficient to rely on the certification and trust established in A (for encryption) or B (for signatures) when using proxy key $\pi_{A\to B}$, since a valid proxy key can by definition only be generated with the cooperation of the owner. Furthermore, the proxy function can be safely applied at any convenient time or place, by the message's sender or receiver, or at any intermediate (and possibly untrusted) point in the network.

Proxy functions potentially also have practical utility for key management in real systems. For example, some pieces of secure hardware (*e.g.,* smartcards) limit the number of secret keys that can be stored in secure memory, while some applications might require the ability to decrypt messages for more keys than the hardware can accommodate. With proxy cryptography, once a new key is created and a corresponding proxy key generated, the secret component of the old (or new) key can be destroyed, with the (public and externally-applied) proxy key maintaining the ability to decrypt for both. In effect, proxy functions allow us to increase the number of public keys without also increasing the number of secret bits or the amount of secret computation. Because proxy functions can be computed anywhere, messaging systems, such as electronic mail, can proxy "forward" messages encrypted with one key to a recipient who holds a different key. Proxy functions make it possible to associate a single key with a network or physical address but still decrypt messages forwarded (and proxied) from other addresses. Finally, proxy functions effectively allow changing or adding a key without obtaining new certificates or altering the distribution channel for the previous public key; this could be useful when it is difficult to distribute or certify new keys (e.g., old keys were published in widely-distributed advertisements or embedded in published software, or the certification authority charges high fees for new certificates).

1.4 Security of proxy schemes and ad hoc substitutes

If Alice wants Bob to be able to read her mail, instead of issuing a proxy key she might just give Bob her secret key (perhaps, obviating the need to involve Bob, by encrypting it in Bob's public key and publishing it). This would be inferior to using a proxy scheme for several reasons. First, as discussed above, Bob's computing environment may be limited and therefore incapable of automatically processing encrypted secret keys; any new software to decrypt and manage such keys would have to run within the environment trusted by Bob. Proxy processing, on the other hand, can take place entirely outside of Alice's and Bob's trusted environments and without their active involvement. Furthermore, encrypting one's secret key with another's public key is not in general secure. The cryptosystem we present below, a variant[1] of ElGamal, is thought to be secure in part because the cryptanalysis problem is random-self-reducibile—which allows one to assert mathematically that recovering m from the public information $\langle e_a, E(m, e_a), e_b\rangle$ is hard on average if it is hard at worst. The task of recovering m from $\langle e_a, E(m, e_a), E(d_a, e_b), e_b\rangle$, however, may be considerably easier since $E(d_a, e_b)$, in the context of e_a and e_b, may leak information about d_a—specifically, the new cryptanalysis problem is probably not random-self-reducible and due to the problem's obscurity it is not clear what, if any, mathematical guarantees of security can be given. By contrast, the proxy scheme we give below is just as strong as the underlying ElGamal-like cryptosystem.[2]

1.5 Related work

A natural question to ask is whether there exist atomic proxy functions (and feasible schemes to generate proxy keys) for any public key cryptosystems.

Previous work on delegating the power to decrypt has focused on developing efficient transformations that allow the original recipient to forward *specific ciphertexts* to another recipient. Mambo and Okamoto[MO97] develop this formulation and give efficient transforms (more efficient than decryption and re-encryption) for ElGamal and RSA. Mambo, Usuda and Okamoto[MUO96] apply a similar notion to signature schemes.

While such schemes have value from the standpoint of efficiency, they are not, however, "atomic proxy cryptosystems" by our definition because the transforming function must be kept secret and applied online by the original keyholder on a message-by-message basis (the schemes are not atomic). The security semantics of these systems are essentially the same as a decryption operation followed by a re-encryption operation for the new recipient. Our formulation of proxy cryptography is distinguished from the previous literature by the ability of the keyholder to publish the proxy function and have it applied by untrusted parties without further involvement by the original keyholder.

2 Proxy encryption

Although the problem of proxy cryptography seems like a natural extension of public-key cryptography, existing cryptosystems do not lend themselves to obvious proxy functions. RSA[RSA78] with a common modulus is an obvious candidate, but that scheme is known to be insecure[Sim83][DeL84]. Similarly, there do not appear to be obvious proxy functions for many of the previous discrete-log-

[1] David Wagner notes that our proxy scheme can be extended to work with standard ElGamal[ElG85] encryption.

[2] Note that Bob of this example may be a government mandating that Alice provide him with access to her key. It has been argued that such a scheme makes the system as a whole less trustworthy due to the extra engineering effort involved; we argue here that in the case of random-self-reducible cryptosystems such as ElGamal variants, requiring Alice to encrypt her secret key using the government's public key may also weaken the underlying cryptosystem in the precise mathematical sense of spoiling the random-self-reducibility.

based cryptosystems. This is not to say, of course, that proxy functions for existing systems do not exist, only that we have not discoverd them.

We now describe a new secure discrete-log-based public-key cryptosystem that does have a simple proxy function. The scheme is similar in structure to ElGamal encryption[ElG85], but with the parameters used differently and the inverse of the secret used to recover the message. (This approach has merit beyond proxy encryption; [Hug94] proposed a Diffie-Hellman-like key agreement protocol based on the inverse of the secret, which allows a message's sender to determine the key prior to identifying its recipient).

2.1 Cryptosystem $\mathcal{X}$ (encryption)

Let p be a prime of the form $2q+1$ for a prime q and let g be a generator in Z_p^*; p and g are global parameters shared by all users. A's secret key a, $0 < a < p-1$, is selected at random and must be in Z_{2q}^*, *i.e.*, relatively prime to $p-1$. (A also calculates the inverse $a^{-1} \bmod 2q$). A publishes the public key $g^a \bmod p$. Message encryption requires a unique randomly-selected secret parameter $k \in Z_{2q}^*$. To encrypt m with A's key, the sender computes and sends two ciphertext values (c_1, c_2):

$$\begin{aligned} c_1 &= mg^k \bmod p \\ c_2 &= (g^a)^k \bmod p \end{aligned}$$

Decryption reverses the process; since

$$c_2^{(a^{-1})} = g^k \pmod{p}$$

it is easy for A (who knows a^{-1}) to calculate g^k and recover m:

$$m = c_1((c_2^{(a^{-1})})^{-1}) \bmod p$$

The speed of the scheme is comparable to standard ElGamal encryption, although initial key generation requires the additional calculation and storage of a^{-1}.

2.2 Symmetric proxy function for $\mathcal{X}$

Observe that the c_1 ciphertext component produced by Cryptosystem $\mathcal{X}$ is independent of the recipient's public key. Recipient A's key is embedded only in the c_2 exponent; it is sufficient for a proxy function to convert ciphertext for A into ciphertext for B to remove A's key a from c_2 and replace it with B's key b. Part of what a proxy function must do, then, is similar to the first step of the decryption function, raising c_2 to a^{-1} to remove a. The proxy function must also contribute a factor of b to the exponent. Clearly, simply rasing c_2 to a^{-1} and then to b would accomplish this, but obviously such a scheme would not qualify as a secure proxy function; anyone who examines the proxy key learns the secret keys for both A and B.

This problem is avoided, of course, by combining the two steps into one. Hence, the proxy key $\pi_{A\to B}$ is $a^{-1}b$ and the proxy function is simply $c_2^{\pi_{A\to B}}$. Note that this is a symmetric proxy function; A and B must trust one another bilaterally. B can learn A's secret (by multiplying the proxy key by b^{-1}), and A can similarly discover B's key. This proxy function is also translucent; the proxy key does not directly reveal A or B, but anyone can verify a guess by encrypting a message with A's public key, applying the proxy function, and comparing the result with the encryption of the same message (with the same k) with B's public key. Observe that applying the proxy function is more efficient than decryption and re-encryption, in that only one exponentiation is required.

2.3 Security of $\mathcal{X}$

First, we show that $\mathcal{X}$ is secure—that cleartext and secret keys cannot be recovered from ciphertext and public keys. Beyond that, we also show that publishing the proxy key compromises neither messages nor secret keys. Since recovering a secret key enables an adversary to recover a message and since cryptanalysis is easier with more information (i.e., a proxy key), it is sufficient to show that no cleartext is recoverable from ciphertext, public keys, and proxy keys. Specifically, we will show that the problem of recovering m from

$$(g^a, g^b, g^c, \ldots, mg^k, g^{ak}, a^{-1}b, a^{-1}c, \ldots).$$

is at least as hard as Diffie-Hellman.

Theorem 1 *Suppose there exists a randomized algorithm f that with probability $\epsilon > 1/|p|^{O(1)}$ succeeds in recovering m from the public information*

$$(g^a, g^b, \ldots, mg^k, g^{ak}, b/a, \ldots)$$

where the probability is taken over f's random choices as well as over m and the parameters a, b, and k. Then, for each $\eta = 2^{-|p|^{O(1)}}$, there exists a randomized polynomial-time algorithm for Diffie-Hellman that succeeds with probability $1 - \eta$.

Proof. For simplicity we assume there are only two public keys and one proxy key; the general case is similar. Suppose we had an algorithm F that always succeeds in recovering m from $(g^a, g^b, mg^k, g^{ak}, b/a)$. Then note that on input g^x, g^y, we have $F_1(g^x, g^y) = F(g^y, g^y, 1, g^x, 1)^{-1} = g^{x/y}$, and since $F_1(g, g^y) = g^{1/y}$ we'd have

$$F_2(g^x, g^y) = F_1(g^x, F_1(g, g^y)) = F_1(g^x, g^{1/y}) = g^{xy}.$$

In fact, f is only guaranteed to recover m with probability ϵ so we need to use the random-self-reducibility [Fei93] of the discrete log to achieve our objective.

On input g^x, g^y for $x, y \in Z_{2q}^*$ let f_1 pick $r, s, t, u \in Z_{2q}^*$ at random and query

$$f(g^{ry}, g^{sy}, g^t, g^{ux}, s/r).$$

By hypothesis, with probability ϵ we get $g^{t-(ux)/(ry)}$ from which f_1 can recover $g^{x/y}$. Since $f_1(g, g^y) = g^{1/y}$ we can define f_2 by

$$f_2(g^x, g^y) = f_1(g^x, f_1(g, g^y));$$

this is equal to

$$f_1(g^x, g^{1/y}) = g^{xy}$$

with probability at least ϵ^2.

Our next step is to construct an algorithm that runs correctly with high probability on most inputs in Z_{2q}^*. For this, we define the algorithm $f_3(g^x, g^y)$, $x, y \in Z_{2q}^*$, as follows. Pick r, s, t, u at random with $r, s, u \in Z_{2q}^*$ and $0 \le t < 2q$ even. Compute $f_2(g^{rx}, g^{sy}) = g^{rsxy+c}$ and $f_2(g^{(t+x)/u}, g^{uy}) = g^{ty+xy+c'}$ for some c, c' that depend on the respective input to f_2. Check whether $(g^{rsxy+c})^{1/rs} =$

$(g^{ty+xy+c'})/g^{ty}$ and is of the form g^z for $z \in Z_{2q}^*$ and if so output the common value, otherwise abort.

We need to show that the probability that f_3 outputs the correct answer is substantial and the probability that it outputs an incorrect answer is negligible. Note that the inputs to f_2 are random, so, by hypothesis, at least ϵ^4 of the time $c = c' = 0$ and therefore f_3 will answer correctly. For f_3 to answer incorrectly, we must have $c, c' \neq 0$ and $c/rs = c'$. Note also that in this case c and c' must be even so that $xy + c' = xy + c/(rs)$ are in Z_{2q}^*. Even conditioned on the four inputs to two calls to f_2, the six random variables r, s, t, u, x, y have two degrees of freedom left, and it is easy to see that r and s and therefore rs remain uniformly distributed. Thus, c/rs is uniformly distributed among even numbers modulo $2q$ and only equals c' with probability $1/q$. Thus if we repeat f_3 a total of $O(-\log \eta)/\epsilon^4$ times we will have a probability $1 - \eta$ of a correct answer and only a tiny chance of getting any incorrect answer.

Next, we show how to construct an algorithm $f_4(g^x, g^y)$ that succeeds, with high probability, on all inputs g^x, g^y such that $x, y \in Z_{2q}^*$. Pick r, s at random from Z_{2q}^* and compute $f_3(g^{rx}, g^{sy})^{1/rs}$. The input to f_3 is uniformly distributed, so by hypothesis $f_3(g^{rx}, g^{sy}) = g^{rsxy}$ with high probability and we recover g^{xy}.

Before considering general g^x, g^y we recall some facts about arithmetic modulo $2q$. The integers modulo $2q$ consist of 0, q, $(q-1)$ multiples of 2 (other than 0), and $(q-1)$ invertible elements (the odd numbers other than q). Given an input g^x where g is a primitive element modulo $p = 2q+1$, one regards x modulo $2q$. We can learn whether x is invertible from g^x: If $x = 0$ then $g^x = 1$, if $x = q$ then $g^x = -1$, if x is odd then $(g^x)^q = g^q = -1$ and if x is even then $(g^x)^q = g^0 = 1$. (Raising g^x to the power q is polynomial-time, but expensive. However, we do not need to do this when using the cryptosystem.)

Finally, consider general input g^x, g^y. The cases $x = 0, x = q$ or $y = 0, y = q$ are easy to detect and handle, so we assume that we are not in one of these cases. We can determine s and t in $\{0, 1\}$ so that $x + sq, y + tq \in Z_{2q}^*$. We have $g^{xy} = g^{(x+sq)(y+tq)}/g^{xtq+ysq+stq^2} = \pm g^{(x+sq)(y+tq)} = \pm f_4(g^{x+sq}, g^{y+tq})$ (and we can determine the sign). $\square$

Similarly one can show that recovering a from $(g^a, g^b, mg^k, g^{ak}, b/a)$ is as hard as the discrete log, so publishing the proxy key does not compromise a—not even to the level of Diffie-Hellman.

3 Proxy identification

In this section we describe a discrete-log-based identification scheme. With p, g, a as before, Alice wishes to convince Charlotte that she controls a; Charlotte will verify using public key g^a. As before, the proxy key $\pi_{A \to B}$ will be a/b—it will be safe to publish a/b and Alice and Charlotte can easily use a/b to transform the protocol so Charlotte is convinced that Alice controls b.

Note that in the case of a secure identification proxy key that transforms identification by A into identification by B, it is B whose secret is required to construct the proxy key because identification as B should not be possible without B's cooperation.

3.1 Cryptosystem $\mathcal{Y}$ (identification)

Let p and g be a prime and a generator in Z_p^*, respectively. Alice picks random $a \in Z_{2q}^*$ to be her secret key and publishes g^a as her public key. Each round of the identification protocol is as follows:

- Alice picks a random $k \in Z_{2q}^*$ and sends Charlotte $s_1 = g^k$.
- Charlotte picks a random bit and sends it to Alice.
- Depending on the bit received, Alice sends Charlotte either $s_2 = k$ or $s_2' = k/a$.
- Depending on the bit, Charlotte checks that $(g^a)^{s_2'} = s_1$ or that $g^{s_2} = g^k$.

This round is repeated as desired. As with existing protocols, there may be ways to perform several rounds in parallel for efficiency [FFS88].

3.2 Symmetric proxy function for $\mathcal{Y}$

A symmetric proxy key is simply a/b. Proxy identification is useful as follows: Suppose Charlotte wants to run the protocol with g^b instead of g^a. Either Alice or Charlotte or any intermediary can use the proxy key to convert Alice's responses k/a to k/b. Also, either party can transform its share of the key pair (a, g^a) to b or g^b before any protocol takes place. Thus Alice and Bob can authenticate for each other but otherwise the protocol is secure. This proxy scheme is translucent.

3.3 Security of $\mathcal{Y}$

Theorem 2 *Protocol $\mathcal{Y}$, with or without proxy keys published, is a zero-knowledge protocol that convinces the verifier that the prover knows the secret key.*

Proof. Without proxy keys published, this protocol is similar to others in the literature (see, e.g., [FFS88]). Note that if a prover could produce both k/a and k then the prover could produce a from g^a (perhaps only with significant probability).

Now suppose that a proxy key a/b is published for random public keys g^a and g^b, and suppose that D can then impersonate A. Since D could already generate a random proxy key r and matching public key g^{ar}, it follows that D could impersonate A even without knowing a/b and g^b. Thus publishing proxy keys does not weaken the system. $\square$

4 Proxy signature

The concept of proxy cryptography also extends to digital signature schemes. A signature proxy function transforms a message signature so that it will verify with a public key other than that of the original signer. In other words, a signature proxy function $\Pi(s, \pi_{A \to B})$ with proxy key $\pi_{A \to B}$ transforms signature s signed by the secret component of key A such that $V(m, \Pi(S(m, A), \pi_{A \to B}), B) =$ VALID, where $S(m, k)$ is the signature function for message m by key k and $V(m, s, k)$ is the verify function for message m with signature s by key k.

Again, existing digital signature schemes such as RSA[RSA78], DSA[NIS91], ElGamal[ElG85], etc. do not have obvious proxy functions (which, again, is not to say that such functions do not exist).

As in the case of proxy identification, in order to construct a proxy key that transforms A's signature into B's signature, B's secret must be required to construct the proxy key because signing for B should not be possible without B's cooperation.

Now we will see how to use the proxy identification scheme to construct a proxy signature scheme. We suppose there exists a hash function h whose exact security requirements will be discussed below. The parameters p, g, a, b are as before.

4.1 Cryptosystem $\mathcal{Z}$ (signature)

To sign a message m, Alice picks $k_1, k_2, \ldots k_\ell$ at random and computes $g^{k_1}, \ldots g^{k_\ell}$. Next Alice computes $h(g^{k_1}, \ldots g^{k_\ell})$ and extracts ℓ pseudorandom bits $\beta_1, \ldots, \beta_\ell$. For each i, depending on the i'th pseudorandom bit, Alice (who knows a) computes $s_{2,i} = (k_i - m\beta_i)/a$; that is, $s_{2,i} = (k_i - m)/a$ or $s_{2,i} = k_i/a$. The signature consists of two components:

$$\begin{aligned} s_1 &= (g^{k_1}, \ldots, g^{k_\ell}) \\ s_2 &= ((k_1 - m\beta_1)/a, \ldots, (k_\ell - m\beta_\ell)/a) \end{aligned}$$

To verify the signature, first the β_i's are recovered using the hash function. The signature is then verified one "round" at a time, where the i'th round is $(g^{k_i}, (k_i - m\beta_i)/a)$. To verify $(g^k, (k - m\beta)/a)$ using public key g^a, the recipient Charlotte raises (g^a) to the power $(k - m\beta)/a$ and checks that it matches $g^k/g^{m\beta}$.

4.2 Symmetric proxy function for $\mathcal{Z}$

A symmetric proxy key $\pi_{A\to B}$ for this signature scheme is a/b. The proxy function Π leaves s_1 alone and maps each component $s_{2,i}$ to $s_{2,i}\pi_{A\to B}$. The proxy scheme is translucent.

4.3 Security of $\mathcal{Z}$

This scheme relies on the existence of a "hash" function h. Specifically,

Assumption 1 *We assume there exists a function h such that:*

- *On random input (g^a, m), it is difficult to generate $\{r_i\}$ and $\{\beta_i\}$ such that*

$$h(g^{ar_1+m\beta_1}, \ldots, g^{ar_\ell+m\beta_\ell}) = \langle \beta_1 \ldots, \beta_\ell \rangle.$$

- *More generally, it is difficult to generate such $\{r_i\}$ and $\{\beta_i\}$ on input g^a, m, and samples of signatures on random messages signed with a.*

It is not our intention to conjecture about the existence of such functions h. In particular, we do not know the relationship between Assumption 1 and assumptions about collision freedom or hardness to invert.[3] We note that this generic transformation of a protocol to a signature scheme has appeared in the literature [FS86].

[3] Assumption 1 *does* imply that, on random input g^a, it is hard to find $\langle r_i \rangle$ making all the β_i's zero, i.e., such that $h(g^{ar_1}, \ldots, g^{ar_\ell}) = 0$.

We now analyze Assumption 1. Note that in order to produce a legitimate signature on m that verifies with g^a, a signer needs to produce $\langle g^{k_i}\rangle$ and $\langle (k_i - m\beta_i)/a\rangle$. Thus, putting $\langle \beta_i\rangle = h(\langle g^{k_i}\rangle)$ and then $\langle r_i\rangle = \langle (k_i - m\beta_i)/a\rangle$, it is straightforward to see that the signer could actually produce r_i's and β_i's of the stated type in the course of producing the signature.

While we do not address the security of h, we can state that issuing proxy keys does not weaken the system.

Theorem 3 *Suppose h satisfies Assumption 1. Then, for most b, it is also hard to produce $\{r_i\}$ and $\{\beta_i\}$ given additional input $a/b, g^b$, and samples of messages signed with b.*

Proof. As above, a signer not having access to b's messages and proxy keys could simulate this by choosing a random proxy key r, generating $g^b = g^{ar}$, and convert some messages signed with a into messages signed with b. $\square$

5 Conclusions

Intuitively, atomic proxy cryptography is a fairly natural extension of the basic notion of public-key cryptography. It surely seems plausible, given that there exist cryptosystems that can grant the ability to encrypt without granting the ability to decrypt, that there might also exist cryptosystems that can grant the ability to *re-encrypt* without granting the ability to decrypt. However, it is not at all obvious whether there exist atomic proxy schemes in general.

Indeed, while this paper has demonstrated that there do exist efficient and secure public-key encryption and signature schemes with symmetric atomic proxy functions, this observation probably raises more new questions than it answers. In particular, do proxy functions exist for public-key cryptosystems based on problems other than discrete-log? (One possibility is that, for some cryptosystems, proxy functions do exist but it is infeasible to find a proxy key.) More importantly, we have yet to discover a secure *asymmetric* proxy function of any kind; asymmetric proxy functions are probably much more useful in practice, since there are likely many situations where trust is only unidirectional. Are there cryptosystems for which asymmetric proxy functions exist?

6 Acknowledgements

The authors thank Steve Bellovin, Jack Lacy, Dave Maher, Andrew Odlyzko and David Wagner for helpful discussions and comments on earlier drafts.

References

[DeL84] J. M. DeLaurentis. A further weakness in the common modulus protocol for the RSA cryptoalgorithm. *Cryptologia*, 8:235–239, 1984.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. IT-31(4):469–472, July 1985.

[Fei93] Joan Feigenbaum. Locally random reductions in interactive complexity theory. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 13:73–98, 1993.

[FFS88] U. Feige, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Crypto 86*, number 263 in LNCS, pages 186–194, Santa Barbara, CA, USA, August 1986.

[Hug94] Eric Hughes. An encrypted key transmission protocol. *CRYPTO '94* Rump Session presentation, August 1994.

[MO97] M. Mambo and E. Okamoto. Proxy cryptosystems: delegation of the power to decrypt ciphertexts. *IEICE Trans. Fundamentals*, E80-A(1), 1997.

[MUO96] M. Mambo, K. Usuda, and E. Okamoto. Proxy signatures: delegation of the power to sign messages. *IEICE Trans. Fundamentals*, E79-A(9), 1996.

[NIS91] NIST. A proposed federal information processing standard for digital signature standard (DSS). Draft Tech. Rep. FIPS PUB XXX, August 1991.

[RSA78] Ron L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Sim83] G. J. Simmons. A "weak" privacy protocol using the RSA crypto algorithm. *Cryptologia*, 7:180–182, 1983.