Proprietary Certificates (Extended Abstract)

Markus Jakobsson[1], Ari Juels[1], and Phong Q. Nguyen[2]

[1] RSA Laboratories
{mjakobsson,ajuels}@rsasecurity.com
[2] CNRS/École normale supérieure
pnguyen@ens.fr

Abstract. Certificates play an essential role in public-key cryptography, and are likely to become a cornerstone of commerce-related applications. Traditional certificates, however, are not secure against *certificate lending*, i.e., a situation in which a certificate holder voluntarily shares with others the rights bestowed upon him through a certificate. This type of abuse is a concern in several types of applications, such as those related to digital rights management.
In this paper, we introduce the notion of *proprietary* and *collateral* certificates. We present a scheme whereby one certificate, known as a *proprietary* certificate, may be linked to another, known as a *collateral* certificate. If the owner of the proprietary certificate shares the associated private key, then the private key of the collateral certificate is simultaneously divulged.
Certificates in our scheme can be integrated easily into standard PKI models and work with both RSA and discrete-log-based keys (such as those for DSS). Our scheme leaks no significant information about private keys, and leaks only a small amount of information about certificate ownership. Thus, use of proprietary certificates still allows users to maintain multiple, unlinkable pseudonyms, and adds functionality without posing any threats to user privacy.

Keywords: certificates, collateral key, digital rights, fair encryption, proprietary key

1 Introduction

A digital certificate assigns an identity or a right to its holder, that is, to the possessor of the associated private key. This assignment is made by way of a digital signature that a certificate authority (CA) applies to the corresponding public key and to a description of the certificate's scope of use. Certificates may be employed for such broad purposes as binding a user identity to a public key for purposes of encryption, or to assign an entity the authority to sign legal documents on its own behalf. More specific situations in which certificates may be useful include the granting of access rights to a building or to a subscription-based service.

It is implicitly assumed that certificates, and the rights that come with them, belong solely to the person or entity to which they were issued. The issue of *non-transferability* of certificates and their associated rights, however, has been only superficially addressed in the literature. In this paper we propose the notion of *proprietary certificates*, which are certificates with the property that their rights cannot be transferred (corresponding to giving somebody the private keys associated with the certified public keys) without punitive leakage of collateral information.

We believe that proprietary certificates may be important for three reasons. One is the likely future dependence on certificates for applications relating to commerce, and the benefit of incorporating new functionality into certificates. A second is the need for user privacy, which our solution preserves to a high degree. The third and perhaps most critical reason is the likely possibility of a proliferation of fair charging and access control mechanisms for information-based Web services in the near future. With many forms of Web advertising in decline (see, e.g., [19]), content providers have expressed a growing need to turn to subscription fees for revenue at some point.

We therefore see the main contribution of our paper as the concept of proprietary certificates, with its possible impact on the development of new services. Our primary focus is to exhibit secure and reasonably efficient structures for proprietary certificates. In doing so, we rely to a large extent on a combination of cryptographic components introduced for other purposes. Our current proposal causes an increase of the certificate size of between 384 and 768 Bytes, depending on the cryptosystems used. Alternatively, this certificate augmentation may be kept externally, indexed by the proprietary certificate it relates to. Such an approach would allow the certificates to retain their exact format while extending their functionality by means of this external record. We note that a further study of appropriate mechanisms – with a focus on their use in proprietary certificates – may result in more compact certificates.

We develop a mechanism to ensure that users can produce multiple unlinkable certified pseudonyms, with the certificates issued by one or many certificate authorities, such that it is impossible for a user to give away the right (to another user) to issue one or more types of signatures or other secret functions related to the certificates. This holds unless the user gives away the right to issue all kinds of signatures for all kinds of certificates and pseudonyms he holds, which means a total impersonation of the "lender". Also, it is possible to produce a system in which some keys (but not all) are released, were some keys to be given out. Thus, if a user is not willing to give away the right to sell his home to a second user, or to sign other legal documents for the first user, he can also not give away or share the right to access a subscription, or to enter a building, etc. (This assumes off-line collaboration, which is a good model for many scenarii.) In other words, we show how we can construct certificates on unlinkable pseudonyms such that the disclosure of one private key (which we will call the *proprietary* key) automatically implies the disclosure of a second private key (what we call the *collateral* key). If two keys are each others' collateral (directly or indirectly), we call the relationship *symmetric*; otherwise *asymmetric*.

A trivial approach would be to have the user employ the same private and public keys for every certificate (or for many). This approach has several drawbacks. First, it immediately and publicly links the identity of the holder to all of the associated certificates, thereby undermining the privacy guarantees afforded by unlinkability. Second, this approach is crude in that it does not permit the establishment of asymmetric relationships. In the trivial approach, disclosure of private keys is all or nothing, whereas our approach allows for a great deal more refinement, as we show below. Finally, in the trivial approach, there is no clear way to link certificates employing different cryptosystems. In contrast, we demonstrate in this paper how to offer this kind of flexibility.

Technically, this can be achieved by incorporating a ciphertext corersponding to the collateral private key (or a represenation thereof), either in the proprietary certificate, or in an external database. (Onwards, we assume the ciphertext to be part of the certificate, for simplicity, but note that the options are technically equivalent.) Given that the encryption of

the collateral key would be performed using the proprietary public key, we have that a party with knowledge of the proprietary private key will be able to derive the collateral private key by decryption of this ciphertext. Assuming the use of semantically secure encryption, the ciphertext will not reveal any information about the link between the two public keys or their certificates. It is important to note that it is not sufficient simply to encrypt one private key using another public key and incorporate the result in the certificate. Namely, it is important to guarantee robustness (i.e., to allow the CA to be certain about the contents of the ciphertext) without having to give the encrypted private key to any party or set of parties. Another technical difficulty is to provide the above functionality for schemes supported by standards, as opposed to schemes and structures designed solely for the purpose of the paper. While the "interior" of our solution contains schemes that are not currently supported by standards (such as Paillier's encryption scheme), it is important to note that the "exterior" of our solution relates to standard schemes, namely RSA and DSS. This means that any RSA or DSS key can be used as a proprietary or collateral key.

Outline. We begin in section 2 by describing the related work, followed in section 3.1 by an informal description of our goals and a statement of the contributions of the paper. We then describe our technical approach in section 3.2, but keep the discussion on a detail-free and intuitive level. In section 4, we then introduce denotation and outline the structure of our modified certificates, and review the building blocks we will use. In section 5 we then describe our solution in technical detail, using the previously introduced building blocks to describe our protocols. We state the properties of our solution in section 6; Appendix A contains proofs of these claims.

2 Related Work

The notion of non-linkability and independence among signatures arises frequently in the literature on digital payments, while the issue of non-transferability of access rights has been investigated from one perspective by Dwork, Lotspiech, and Naor [9], and from another by Goldreich *et al.* [14]. The combination of the two properties, however, has to our knowledge not been considered yet, and poses interesting technical questions as well as the possibility of new applications.

Our work is conceptually related to the work on *signets* by Dwork, Lotspiech and Naor [9], in which a secret, such as a credit card number, is incorporated in a private key to prevent the latter from being given away. Similarly, our aim is to some extent related to that of digital watermarking, as surveyed in [17]. In a digital watermarking system, identifying information of some kind is embedded in an indelible way in an image so as to discourage illicit copying. In neither of these proposals, though, is the embedded private (the collateral key in our terminology) hidden from the party who wishes to verify that it is there. In contrast, this is precisely what occurs in our solution.

The problem we study is also spiritually related to a problem previously studied by Goldreich *et al.* [14]. In their paper, a user owns a certificate associated with some rights, and wishes to delegate a certain portion of these to himself. This allows him to delegate rights for use on a laptop, with the benefit that if this gets stolen, then the damage is limited to the delegated portion of the rights (and the lost machine.) Thus, their scenario is the following: A user has a primary (long-term) key associated with some personalized access rights, some of which he wishes to delegate to some secondary (and short-term) keys.

If sufficiently many secondary keys are disclosed, then the primary key can be recovered from these, thereby preventing the user from giving away his secondary keys. On the other hand, if few secondary keys are disclosed (fewer than a certain threshold), then the primary key remains secure. If the threshold is set to one (as it is in our scheme), however, then their primary and secondary keys are identical. (In other words, the issue of user privacy, or unlinkability of certificates, is not addressed in [14].) On the other hand, if the threshold is set higher, a corrupt user can give out some keys without any risk. Therefore, their solution – which was not intended for securing intellectual property – is also not very well suited for this task.

The problem we address in this paper is related to that studied by Camenisch and Lysanskaya [4], who produce a credential scheme in which signatures (and other authentication elements) are generated from one and the same private key without being linkable to each other. Additionally, and similar to what is achieved in our scheme, they allow different private keys associated with a user to be tied to each other in a way that prevents users from sharing some private keys without sharing others as a result of this. However, while conceptually related from a birds-eye view, the two results are different on several counts. First, we do not provide unlinkability on a signature-by-signature basis. In our scheme, all signatures associated with one public key can be linked to this public key, as is normal for standard signatures and a standard PKI infrastructure. While this linkability is highly undesirable for, e.g., group signatures (whose very goal is for the opposite to hold), it is desirable in an infrastructure with standard signatures where each public key and all its signatures get associated with its owner. (However, it remains desirable that signatures from *different* public keys remain unlinkable, which we provide.) Another difference between the schemes is that the leakage of one key in their scheme immediately results in the leakage of *all* other keys, while our approach allows a tighter control of the inferrable relations between keys – namely, we can employ any graph of symmetric or asymmetric relationships between nodes / keys. This results in many practical advantages. Furthermore, and more importantly, we allow the linking of private keys for *standard* signature schemes (such as DSS and RSA), while the methods in [4] relate only to a new signature scheme introduced in their paper. It is worthwhile to notice that the employment of our methods to existing signature schemes is not only of potential technical value, but also of practical value in any legacy system (of which digital signatures may be one of the best examples).

Technically speaking, our solution depends most importantly on methods for key escrow, for which similar cryptographic building blocks are employed. Young and Yung [23] recently showed how to obtain a software key escrow system in users provide ciphertexts to certification authorities that permit the recovery of users' private keys. These ciphertexts are encrypted under the public key of an escrow authority. The structure that is used to assure the CA that the ciphertext is of the correct format has been called *fair encryption*. Thus, the encryption key in an escrow application is the public key of the escrow authority, while in our system, it is the public proprietary key. Similarly, the *encrypted* key in a escrow scheme is the user's private key, while it corresponds to the collateral key in our scheme for proprietary certificates.

The fair encryption scheme of [23] could be used for purposes of proprietary certificates. In fact, proprietary certificates constitute a new application for fair encryption. In order to allow for compatibility with more common crypto systems – namely RSA and standard discrete-log based schemes – we do not employ their methods, which are based on a "double decker" structure.

We do, however, make direct use of the rather efficient fair encryption scheme of Poupard and Stern [21] for some of the protocols of our scheme, namely those where the proprietary key is an RSA key. We develop and propose new schemes for the case where the proprietary key is a discrete-log key. These new schemes constitute an extension of previous results for fair encryption and are thus of independent interest.

Finally, we employ methods from [12] for proving equality of discrete logs over composite integers. These, in turn, are related to proof methods of Chaum [7] for proving equality of discrete logs over prime-order fields.

3 Goals

3.1 Overview

Just as a person may carry several identifying tokens for access to various resources and rights (such as a driver's license, a passport, and various credit and debit cards), he or she may need several public keys, each one of which may be associated with a different sets of rights. Different public keys (and their certificates) may also be associated with different policy requirements, such as requirements on the methods used to verify the identity of the certificate owner at the time the certificate is issued; the possible escrowing of keys; and the acceptable uses of the certified public key.

Our aim is to construct a proprietary certificate system that respects these requirements for heterogenity and flexibility in a public key infrastructure. We assume that certificate authorities publish directories containing public information on the certificates they have issued. To make our goals precise, let us consider a case in which a certificate authority CA_1 wishes to issue a proprietary certificate C_1 to a certain user. The user is to provide a second certificate C_2, issued by a (possibly) distinct entity CA_2, as collateral. Informally stated, our goals in creating the proprietary/collateral relationship between C_1 and C_2 are as follows:

1. **Non-transferability:** With high probability, any player who learns the private key for C_1 will learn the private key for C_2, and be able to locate public information for C_2 in the directory maintained by CA_2. Thus, given that the user does not wish to relinquish control over C_2, the private keys associated with C_1 are non-transferable.
2. **Unlinkability:** CA_1 learns that the user knows the private key associated with C_2, and that CA_2 issued C_2. CA_1 learns no additional information about certificates held by the user, and no other player learns any information about the certificates of the user.
3. **Cryptosystem agility:** C_1 and C_2 can be based on different cryptosystems: either can make use of an RSA key or a discrete-log key.
4. **Locality:** CA_1 need interact only with the user, and not CA_2.
5. **Security:** CA_1 learns only a negligible amount of information about the private keys associated with C_1 and C_2. No other party learns any information relating to the certificates.
6. **Efficiency:** The certificate C_1 is not substantially larger than a conventional certificate of its type. Moreover, the computational and communication requirements on CA_1 and the user in establishing the proprietary/collateral relationship are reasonable.
7. **PKI compatibility:** We require that the modified certificates allow for easy integration into standard PKI models. (While we require use of so-called "safe" RSA keys, these are fully compatible with most existing mechanisms.)

We may view the collection of certificates belonging to a particular user as a collection of nodes $C = \{C_1, C_2, \ldots, C_t\}$ in a directed graph $G = (C, E)$. An arc $(C_i, C_j) \in E$ in this graph represents a binding of a proprietary certificate C_i to a collateral one C_j. If there is an additional reverse arc (C_j, C_i), then the relationship between C_i and C_j is called *symmetric*; otherwise, it is called *asymmetric*. Nodes may have degrees of arbitrary size. In our system, the size of a certificate C_i is linear in its out-degree, as are the computational and communication requirements to establish outgoing arcs. As an illustration of exactly what purpose an arc serves, we present the following brief example.

Example. Say that a user wants to obtain a certificate for an on-line newsletter. This newsletter requires that the certificate C_i associated with one particular user of their service serve as a proprietary one, with the user's Mastercard certificate C_j as collateral. Moreover, to guard against a situation in which the user closes her Mastercard account, the newsletter may require that the user's public encryption key and corresponding certificate C_k also serve as collateral. (Alternatively, the newsletter may blacklist the certificate C_i once it detects that Mastercard blacklists C_j.) Additionally, Mastercard may employ the user's decryption key as collateral for its own certificate. Thus, the newsletter creates arcs (C_i, C_j) and (C_i, C_k) in the graph G. See figure 1 for a graphical depiction of this scenario.

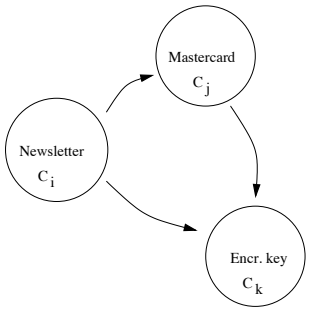

Fig. 1. Example certificate relationships.

Privacy note. It is important to note that the CA, while not learning either the collateral or proprietary secrey key, learns the association between the two public keys. Namely, he learns that the public key of the proprietary certificate is associated with the same person as the public key of the collateral certificate. For many purposes, this may not be so outlandish, as long as the public cannot infer the same relation. To hide the association of keys from the CA, it is possible to use more heavy-weight protocols, in which the user proves correct encryption with respect to an *unspecified* public key belonging to some set of potential collateral public keys. The likely drawback of such solutions, though, would be the resulting reduction of the efficiency. Alternatively, if we *do* allow the CA to learn the association, standard techniques [16] can be employed to prevent him from convincing others.

3.2 What does an arc look like?

Our approach is to include in a proprietary certificate a ciphertext on the collateral keys. This ciphertext (while not necessarily of a standard format) may be decrypted using the public keys of the proprietary private keys, thereby yielding the collateral keys. In order to reach this goal, we need secure protocols for a certificate holder to prove to the CA that the generated ciphertexts are of the right format (namely that they contain valid representations of the collateral private keys). This must be done in a manner that is both efficient and which limits leaks of information to the CA. This will be achieved by basic fair encryption techniques, including use of zero-knowledge proofs and semantically secure encryption. Apart from including an encryption of the private collateral key in the certificate, the certificate authority may additionally include a pointer to the directory entry for the collateral key. For reasons of privacy, this would also be encrypted, using a semantically secure encryption scheme and the public key of the proprietary certificate (making it possible to decrypt given the private key associated with the proprietary certificate). Technically, the encryption of the pointer is straightforward, as the plaintext information does not need to be hidden from the CA issuing the proprietary certificate. Thus, we will focus on the encryption of the private key instead of that of the public key. Practically, it is worth mentioning, though, as it allows the retrieval of private keys as well as an *understanding of what was retrieved*, should the proprietary key be given away.

In a fair encryption system, a user holds a private/public key pair (PK, SK), and a public key PK_T is published for some trusted third party. The user constructs a ciphertext $\Gamma_{1\to 2}$ and a non-interactive proof that $\Gamma_{1\to 2}$ is an encryption under PK_T of a representation of SK or data that enable efficient reconstruction of SK. In our system, $\Gamma_{1\to 2}$ is a ciphertext on the private key for C_2 (or something equivalent) under the public key associated with C_1. A critical difference in our system from conventional use of fair encryption is our assumption that CA_1 is responsible for ensuring that Γ is correctly constructed. (This is the case in all of the applications we envisage, as it is CA_1 that wishes to prevent abuse of C_1.) Hence, the owner of C_2 must prove correct construction of Γ only once to CA_1. In consequence, the proof may be interactive, and the size of the proof is of less importance than in a typical fair encryption system, as it has no impact on the size of C_1, which only carries $\Gamma_{1\to 2}$.

In section 5, we detail the various protocols for creating an arc (C_1, C_2) between two certificates C_1 and C_2. As explained above, we use the Poupard-Stern fair encryption system as the basis for protocols in which C_1 is an RSA-key-based certificate and C_2 is either RSA or DL-based. An important contribution of our paper is a pair of novel fair encryption protocols for the case where C_1 is instead a discrete-log-based certificate and C_2 is either RSA or discrete-log based.

4 Notation and Building Blocks

4.1 Notation

We define a cryptosystem CR in the broadest sense to include a suite of five algorithms $\mathsf{keygen}_{CR}, \mathsf{sign}_{CR}, \mathsf{verify}_{CR}, \mathsf{encrypt}_{CR}$ and $\mathsf{decrypt}_{CR}$ for the respective operations of key generation, signing, verification, encryption, and decryption. (Thus, where we consider a signing algorithm such as, e.g., DSS, we assume a corresponding encryption/decryption algorithm over the same algebraic structures, e.g., El Gamal.) We assume implicitly that

a secure suite of algorithms of this kind are available for all cryptosystems under consideration. To produce a certificate C_1 on a public/private key pair (PK_1, SK_1), a given certificate authority applies an algorithm sign_{CR} to PK_1 and possibly to some additional policy information aux_1.

Our contribution in this paper is a set of protocols for arc creation, and a corresponding set of protocols for key extraction, i.e., for computation of a collateral key given the corresponding proprietary one.

Arc creation: Let (PK_1, SK_1) and (PK_2, SK_2) be the public/private key pairs respectively for proprietary certificate C_1 and collateral certificate C_2. Additionally, let C_1 be a certificate on a public key for cryptosystem type CR_1 and C_2 a certificate for cryptosystem type CR_2. We let $\mathsf{Arc}_{CR_1 \to CR_2}$ denote an arc creation protocol on the proprietary/collateral certificate pair (C_1, C_2).

The protocol $\mathsf{Arc}_{CR_1 \to CR_2}$ takes as input from the prover the two public/private key pairs (PK_1, SK_1) and (PK_2, SK_2), and relevant security cryptosystem security parameters. Input from the verifier is a security parameter specifying protocol soundness. The output of the protocol is the pair of public keys (PK_1, PK_2), the collection of security parameters, and a ciphertext $\Gamma_{1 \to 2}$.

We require two security properties on a given algorithm $\mathsf{Arc}_{CR_1 \to CR_2}$:

- **Soundness:** With overwhelming probability over the coin flips of the prover and verifier, the ciphertext $\Gamma_{1 \to 2}$ is well formed. In other words, given input SK_1 and $\Gamma_{1 \to 2}$, the protocol $\mathsf{Extract}_{CR_1 \to CR_2}$ described below yields output SK_2.
- **Privacy:** The full transcript of the protocol is simulable in a computationally indistinguishable manner by a player with knowledge of PK_1 and PK_2 only. (Thus, e.g., the verifier learns no non-negligible information about SK_1 or SK_2.)

We consider two cryptosystem types in this paper, namely RSA and discrete log (DL). In other words, either CR_1 or CR_2 can be a cryptosystem in which the public key is an RSA modulus, e.g., RSA encryption and signing or Paillier encryption with RSA signing, or, alternatively, a cryptosystem based on discrete log, such as (El Gamal / DSS), (El Gamal / Schnorr,) etc. Thus, in our paper, we specify four generic arc creation protocols: $\mathsf{Arc}_{RSA \to DL}$, $\mathsf{Arc}_{RSA \to RSA}$, $\mathsf{Arc}_{DL \to DL}$, and $\mathsf{Arc}_{DL \to DL}$.

Key extraction: Corresponding to each arc creation algorithm $\mathsf{Arc}_{CR_1 \to CR_2}$ is a key extraction algorithm $\mathsf{Extract}_{CR_1 \to CR_2}$. This algorithm takes as input a ciphertext $\Gamma_{1 \to 2}$ and the keys PK_2 and SK_1. If successful, it outputs SK_2. Involved here are some standard decryption operations in combination with additional cryptographic operations such as lattice reduction and gcd algorithms.

4.2 Building Blocks

Discrete-log-based signature schemes. If the signature scheme associated with key pair (PK, SK) is discrete-log based (such as DSS [18] or Schnorr [22]), then $PK = (p, q, g, y)$, for primes p, q, such that $p = kq + 1$, an element g of order q, and a value $y = g^x \bmod p$. Here, the private key $SK = x$ is chosen uniformly at random from $\mathbb{Z}_q$. Example sizes of the parameters are $(|p|, |q|) = (1024, 160)$. We refer to [18, 22] for more details on the schemes.

El Gamal encryption. Let g be a generator of a large subgroup $\mathcal{G}$ of $\mathbb{Z}_n$. Often, the integer n is chosen to be prime, but we will also consider the use of strong RSA moduli, and we assume that all computation is performed modulo n, where applicable. Let s denote the size of the subgroup generated by g, and let $y = g^x$ be the public key used for encryption, where $x \in \mathbb{Z}_s$.

To encrypt a message m, one picks a random element $\alpha \in_u \mathbb{Z}_s$ and computes the pair $(a, b) = (y^\alpha m, g^\alpha)$. (We note that s can be determined by a party who knows the factorization of n, which will not be a restriction in our setting.) To decrypt a ciphertext (a, b), one computes $m = a/b^x$.

It is well-known that the El Gamal scheme is semantically secure under the Decision Diffie-Hellman (DDH) assumption on the subgroup $\mathcal{G}$, and given that messages are chosen from $\mathcal{G}$ (see [1]). If messages are chosen from another set, the ciphertext may leak some information, such as the Jacobi symbol of the message.

Proof of equality of discrete logs. Let $y_i = {g_i}^{x_i}$, for $i \in \{1, 2\}$, where $y_1, y_2, g_1, g_2 \in \mathcal{G}$ for some group $\mathcal{G}$. We let $\mathsf{EQDL}_1(y_1, y_2, g_1, g_2)$ denote a zero-knowledge proof protocol demonstrating that $\log_{g_1} y_1 = \log_{g_2} y_2$. There are many methods proposed in the literature for implementing EQDL_1, [7, 22]. In the appendix, we exhibit a version of [22] modified for use with RSA moduli, and discuss its security.

A useful variant employed in our protocols involves elements spanning two groups $\mathcal{G}_1$ and $\mathcal{G}_2$. In particular, we let $\mathsf{EQDL}_2(y_1, y_2, g_1, g_2, n_1, n_2)$ denote a proof that $y_1 = g_1^{x_1} \bmod n_1$ and $y_2 = g_2^{x_2} \bmod n_2$ for $x_1 = x_2$. (In general, $\mathcal{G}_1$ and $\mathcal{G}_2$ need not be modular multiplicative groups, but these are the only type used here.) We use the very efficient proof technique for EQDL_2 introduced in [5].

Both EQDL_1 and EQDL_2 are zero-knowledge. While the soundness of both protocols relies on the discrete log assumption, we note that the soundness of the efficient, one-round version of the protocol for EQDL_2 depends additionally on the *strong RSA* assumption. See [5] for more detailed discussion.

Let $(a, b) = (my^k, g^k)$ represent an El Gamal ciphertext under public key y. Observe that a prover with knowledge of the private key $x = \log_g y$ can prove in zero-knowledge that (a, b) represents a valid ciphertext on plaintext m simply by proving $\mathsf{EQDL}_1(y, a/m, g, b)$.

Paillier encryption. The Paillier cryptosystem was introduced in [20]. It uses the Carmichael lambda function $\lambda(N)$ defined as the largest order of the elements of $\mathbb{Z}_N^*$. Let $N = PQ$ be an RSA modulus such that $\varphi(N)$ is coprime to N. Recall that $\lambda(N) = \mathsf{lcm}(p - 1, q - 1)$. The general Paillier's cryptosystem, as defined in [20], uses an integer G of order multiple of N modulo N^2. It was noticed in [8, 6] that the simplest choice is probably $G = 1 + N$, because $(1 + N)^M \equiv 1 + MN \bmod N^2$. Thus, in this paper, we only use $G = 1 + N$, which slightly simplifies the description of the scheme and has no impact on the semantic security: we refer to [20] for a general description. The public key is N and the secret key is $\lambda(N)$.

To encrypt a message $M \in \mathbb{Z}_N$, randomly choose $u \in \mathbb{Z}_N^*$ and compute the ciphertext $c = (1 + MN)u^N \bmod N^2$. To decrypt c, compute:

$$M = \frac{L(c^{\lambda(N)} \bmod N^2)}{\lambda(N)} \bmod N,$$

where the L-function takes as input an element congruent to 1 modulo N, and outputs $L(u) = \frac{u-1}{N}$.

The Paillier public-key cryptosystem is semantically secure under the hardness of distinguishing N-th residues modulo N^2 (see [20]).

Fair encryption methods. Assume that users have pairs of public and private keys and give an encryption E of their private key (or something allowing the recovery of the private key) using the public key PK_T of a trusted third party. A *fair encryption* is a publicly verifiable proof that the third party is able to recover the private key using his own private key and the ciphertext. Poupard and Stern proposed practical fair encryption [21] using the Paillier cryptosystem (meaning that the third party's public key is a Paillier public key). Poupard-Stern proposed two protocols: One to encrypt El Gamal-type keys, and one for RSA keys. To the best of our knowledge, no fair encryption protocol in which the third party uses a discrete log system exists in the literature. There exist other fair encryption protocols (see [10, 2] for instance), but they do not seem to be as efficient as the Poupard-Stern protocols for our application, so we do not use them.

We will use fair encryption as a proof that any person knowing the private key corresponding to PK_T can recover the private key encrypted in E. In other words, in the setting of proprietary certificates, the "third party" is any possessor of the proprietary private key, and the fair encryption is the proof of collateral private key recovery.

5 Arc creation protocols

We will now consider how one can perform the various proofs of ciphertext correctness, with the various types of encryption needed. We will denote the various protocols by the types of proprietary and collateral keys they relate to. Thus, a $DL \rightarrow RSA$ protocol is a protocol for proving that given a ciphertext and the correct discrete log private key (the proprietary key), one can decrypt and obtain the correct RSA private key (the collateral key). We note that we will use Paillier's encryption scheme in lieu of RSA – however, since one can perform Paillier encryption and decryption using an RSA public versus private key, this is not a restriction.

In the following, we let (y, x) be a public key / private key pair for a discrete-log-based scheme, as described previously, and (e, d) be the public versus private keys of an RSA scheme with public modulus N. We use the same moduli and generators as previously shown. For $\mathsf{Arc}_{DL \rightarrow RSA}$ and $\mathsf{Arc}_{DL \rightarrow DL}$, it is necessary to include in the certificates a generator G as described below. We note that this does not impact the unlinkability properties, since G relates to the public key in whose certificate it is included.

5.1 RSA → DL

Let (e_1, d_1) denote the proprietary public and private keys corresponding to a public modulus N_1, and (y_2, x_2) the collateral public and private keys, with associated modulus p. His Paillier public key is N_1, and his private key is $\lambda(N_1)$.

In the protocol $\mathsf{Arc}_{RSA \rightarrow DL}$, the user randomly chooses $u \in \mathbb{Z}^*_{N_1}$ and computes the ciphertext $\Gamma_{1 \rightarrow 2} = (1 + x_2 N_1) u^{N_1} \bmod {N_1}^2$, and a non-interactive proof (to the CA) of the "third party"'s ability to compute x_2 from y_2 and $\Gamma_{1 \rightarrow 2}$, using the Poupard-Stern fair encryption of El Gamal keys [21, Sect. 3.1].

Extraction of keys. The algorithm $\mathsf{Extract}_{RSA \to RSA}$ involves application of the key recovery process of the Poupard-Stern fair encryption [21, Proof of Theorem 1], based on Gauss lattice reduction algorithm (note: a simple Paillier decryption presumably does not always enable to recover the private key, due to some cheating strategy, as explained in [21]; the proof refers to this key recovery process and not Paillier decryption). This yields x_2.

5.2 RSA → RSA

Let (e_1, d_1) denote the proprietary public and private keys associated with a public modulus N_1, and (e_2, d_2) the collateral public and private keys associated with a public modulus N_2. His Paillier public key is N_1, and his private key is $\lambda(N_1)$.

In the protocol $\mathsf{Arc}_{RSA \to RSA}$, the user computes $x = N_2 - \varphi(N_2)$, randomly chooses $u \in \mathbb{Z}^*_{N_1}$ and the ciphertext $\Gamma = (1 + xN_1)u^{N_1} \bmod N_1^2$. He proves to the CA that a party with knowledge of the decryption key (i.e., our proprietary key) is able to factor N_2 using $\Gamma_{1 \to 2}$ and his Paillier private key, using the Poupard-Stern fair encryption of RSA keys [21, Sect 3.2].

Extraction of keys. To recover the collateral private key using $\mathsf{Extract}_{RSA \to RSA}$, one must apply the key recovery process of the Poupard-Stern fair encryption [21, Proof of Theorem 2] to obtain the factorization of N_2 from Γ and the Paillier private key.

5.3 DL → RSA

Let (y_1, x_1) be the public/private key pair for the DL (i.e., proprietary) certificate, and let N_2 be the modulus for the RSA (i.e., collateral) certificate. For the user to ensure privacy of her private keys, we require that N be the product of two safe primes. Namely, we should have $N_2 = PQ$ where P, Q, $(P-1)/2$ and $(Q-1)/2$ are all large primes. (thus, in particular, P and Q are congruent to 3 modulo 4). The use of safe primes can be proved using [5]. To ensure soundness of the protocol, however, the user need only prove about N_2 that it is the product of (at most) two primes. This can be accomplished with practical computational and communication requirements by combining protocols from [11] or [15] with those in [3], as shown in, e.g., [13].

Apart from a proof that N_2 is a well-formed RSA modulus, there are two key components to the protocol $\mathsf{Arc}_{DL \to RSA}$. The first is that of *key translation*. This is a procedure whereby the user constructs a generator G with large order in $\mathbb{Z}_{N_2}$ and a public El Gamal key $Y = G^{x_1} \bmod N_2$. Since x_1 is the private key for the DL certificate of the user, a player with access to this private key will be able to decrypt any El Gamal ciphertext under public key[1] Y. The user proves correct translation through straightforward use of EQDL_2.

The second key component in the protocol is encryption of a non-trivial root r of unity in $\mathbb{Z}^*_{N_2}$. In particular, the user constructs an El Gamal ciphertext (a, b) on such a root r under the public key Y. Given r, it is easy to compute a factor of N_2, and thus compute any private key for the RSA certificate (provided that N_2 is a well-formed RSA modulus). To prove that the plaintext r corresponding to (a, b) is indeed a root of unity, the user must prove that (a^2, b^2) has plaintext 1. To see that r is a non-trivial root, i.e., not equal to 1 or -1, the CA must verify the following three Jacobi quantities:

[1] Note that this public key will have order at least (P-1)(Q-1)/4 with overwhelming probability. Thus, with overwhelming probability, the choice of public key will itself leak no information about the plaintext root.

– The integer -1 has Jacobi symbol 1. (This is always the case if N is a product of two large safe primes.)
– The value b has Jacobi symbol 1.
– The value a has Jacobi symbol -1.

Together, these three checks ensure that (a, b) has a plaintext r with Jacobi symbol -1, and thus that $r \notin \{-1, 1\}$ and is thus non-trivial. With all of the other proofs given above, this ensures that a player with knowledge of x can use the ciphertext (a, b) to factor N_2 and obtain any private keys associated with the RSA certificate.

Here is our protocol in detail. If any of the verification performed by the CA fails, or any of the sub-protocols fails, then the protocol is aborted.

Protocol $\mathsf{Arc}_{DL \to RSA}$

1. The user selects an element $G \in \mathbb{Z}^*_{N_2}$ of Jacobi symbol 1 such that $G^2 \not\equiv 1$ and $G^2 - 1$ is coprime to N_2. Thus, G has multiplicative order of either $(P-1)(Q-1)/4$ or $(P-1)(Q-1)/2$ (see for instance [12]). Note that the DDH problem in the subgroup spanned by G is believed to be hard (see [1]). The user sends G to the CA.
2. The user performs the key translation. She computes $Y = G^{x_1} \bmod N_2$ and proves $\mathsf{EQDL}_2[g, y, G, Y, n, N]$.
3. The user computes a non-trivial root r of unity with Jacobi symbol -1. This is easy to accomplish given knowledge of P and Q and use of the Chinese Remainder Theorem.
4. The user selects an encryption factor $\alpha \in \mathbb{Z}_{(P-1)(Q-1)/2}$ uniformly at random. She constructs an El Gamal ciphertext on r of the form $(a, b) = (Y^\alpha r, G^\alpha)$. She sends this to the CA.
5. The user proves that (a, b) is a ciphertext under Y of a root of unity. In particular, she proves $\mathsf{EQDL}_1[G, Y, b^2, a^2]$.
6. CA verifies that -1 and b have Jacobi symbol 1, and that a has Jacobi symbol -1.

Key extraction. The algorithm $\mathsf{Extract}_{DL \to RSA}$ interprets the proprietary key x_1 as a key $X = x_1$ for the composite El Gamal ciphertext $E = (a, b)$, one can compute $r = a/b^X \bmod N_2$. One derives the factorization of N_2 by simple gcd: Indeed, $r^2 = 1 \bmod N_2$ implies $(r-1)(r+1) = 0 \bmod N_2$ where $r \neq \pm 1 \bmod N_2$, so that $\gcd(r-1, N_2)$ is a non-trivial factor of N. This yields the private collateral key.

5.4 DL → DL

In order to use a discrete-log proprietary key and a discrete-log collateral key – although possibly over different group structures, we introduce the use of *intermediary keys.* This is a key whose only use is to act as a connector between existing protocols for putting up collateral and performing extraction. Namely, when performing extraction, the proprietary key is used to obtain the intermediary key (serving as a collateral), and then the intermediary key is used as proprietary key to obtain the real collateral key.

Thus, in the protocol $\mathsf{Arc}_{DL \to DL}$, the user selects a strong RSA modulus N' as an intermediary public key (whose corresponding private key is $\varphi(N')$). He then uses the $DL \to RSA$ protocol above to establish N' as the collateral key of his proprietary key. Then, he uses N' as the proprietary key in a $RSA \to DL$ protocol (Poupard/Stern).

The result is two sets of ciphertexts, one containing the intermediary key, and using the proprietary key for encryption/decryption; the second containing the collateral key, and using the intermediary key for encryption/decryption.

Key extraction. The protocol $\mathsf{Extract}_{DL \to DL}$ is an obvious composition of the previous key extraction protocols $\mathsf{Extract}_{DL \to RSA}$ and $\mathsf{Extract}_{RSA \to DL}$. This is a two-step process in which one first obtains the intermediary key and then the collateral key.

6 Claims

We prove in appendix A that our solution satisfies *non-transferability*, *unlinkability* and *security*. It is clear from our protocol description that it satisfies *cryptosystem agility*; *locality*; and *PKI compatibility*. We address the efficiency of our scheme in appendix A as well.

References

1. D. Boneh. The decision Diffie-Hellman problem. In *Proc. of ANTS-III*, volume 1423 of *LNCS*, pages 48–63. Springer-Verlag, 1998.
2. F. Boudot and J. Traoré. Efficient publicly verifiable secret sharing schemes with fast or delayed recovery. In *ICICS '99*, volume 1726 of *LNCS*, pages 87–102. Springer-Verlag, 1999.
3. J. Boyar, K. Friedl, and C. Lund. Practical zero-knowledge proofs: Giving hints and using deficiencies. *Journal of Cryptology*, 4(3):185–206, 1991.
4. J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In B. Pfitzmann, editor, *Eurocrypt '01*, volume 2045 of *LNCS*, pages 93–117. Springer-Verlag, 2001.
5. J. Camenisch and M. Michels. Seperability and efficiency for generic group signature schemes. In M. Wiener, editor, *Crypto '99*, volume 1666 of *LNCS*, pages 413–430. Springer-Verlag, 1999.
6. D. Catalano, R. Gennaro, N. Howgrave-Graham, and P. Q. Nguyen. Paillier's cryptosystem revisited. In P. Samarati, editor, *8th ACM Conference on Computer and Communications Security*. ACM Press, 2001. To appear.
7. D. Chaum and H. Van Antwerpen. Undeniable signatures. In G. Brassard, editor, *Crypto '89*, volume 435 of *LNCS*, pages 212–216. Springer-Verlag, 1989.
8. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *PKC '01*, volume 1992 of *LNCS*, pages 119–136. Springer-Verlag, 2001.
9. C. Dwork, J. Lotspiech, and M. Naor. Digital signets: Self-enforcing protection of digital information. In *Proc. of 28th STOC*, pages 489–498. ACM, 1996.
10. E. Fujisaki and T. Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In K. Nyberg, editor, *Eurocrypt '98*, volume 1403 of *LNCS*, pages 32–46. Springer-Verlag, 1998.
11. Z. Galil, S. Haber, and M. Yung. Minimum-knowledge interactive proofs for decision problems. *Siam J. of Computing*, 18(4):711–739, 1989.
12. R. Gennaro, H. Krawczyk, and T. Rabin. RSA-based undeniable signatures. In B. Kaliski, editor, *Crypto '97*, volume 1294 of *LNCS*, pages 132–149. Springer-Verlag, 1997.
13. R. Gennaro, D. Micciancio, and T. Rabin. An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In *5th ACM Conference on Computer and Communications Security*, pages 67–72. ACM Press, 1998.
14. O. Goldreich, B. Pfitzmann, and R. L. Rivest. Self-delegation with controlled propagation - or what if you lose your laptop. In H. Krawczyk, editor, *Crypto '98*, volume 1462 of *LNCS*, pages 153–168. Springer-Verlag, 1998.

15. J. van de Graaf and R. Peralta. A simple and secure way to show the validity of your public key. In B. Kaliski, editor, *Crypto '87*, volume 293 of *LNCS*, pages 128–134. Springer-Verlag, 1987.
16. M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In U. Maurer, editor, *Eurocrypt '96*, LNCS, pages 143–154. Springer-Verlag, 1996.
17. S. Katezenbeisser and F.A.P. Petitcolas, editors. *Information Hiding Techniques for Steganography and Digital Watermarking*. Artech House, 1999.
18. National Institute of Standards and Technology (NIST). *FIPS Publication 186: Digital Signature Standard*, May 1994.
19. Bloomberg News. Ad-revenue worries weigh down Yahoo. 1 September 2000. URL: http://yahoo.cnet.com/news/0-1005-200-2670551.html.
20. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Eurocrypt '99*, volume 1592 of *LNCS*, pages 223–238. Springer-Verlag, 1999.
21. G. Poupard and J. Stern. Fair encryption of RSA keys. In B. Preneel, editor, *Eurocrypt '00*, volume 1807 of *LNCS*, pages 173–190. Springer-Verlag, 2000.
22. C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991.
23. A. Young and M. Yung. Auto-recoverable auto-certifiable cryptosystems. In K. Nyberg, editor, *Eurocrypt '98*, volume 1403 of *LNCS*, pages 17–31. Springer-Verlag, 1998.

A Analysis

Non-transferability. The scheme satisfies non-transferability if the CA can be guaranteed that for any certificate he has issued, knowledge of its proprietary private key allows the corresponding collateral private key to be computed with an overwhelming probability, and in polynomial time. Thus, this directly corresponds to the soundness of the protocol for proving that the ciphertext in an encryption of the appropriate plaintext (the collateral private key, or a representation thereof), and under the appropriate public key (the proprietary public key.)

The soundness of the fair encryption schemes used for arc $\mathsf{Arc}_{RSA \to DL}$ and $\mathsf{Arc}_{RSA \to RSA}$ has already been proven (see [21]). Since $\mathsf{Arc}_{DL \to DL}$ is composed of $\mathsf{Arc}_{DL \to RSA}$ and $\mathsf{Arc}_{RSA \to DL}$ (the latter which we know is sound), we see that only the soundness of $\mathsf{Arc}_{DL \to RSA}$ remains to be proven.

The soundness of EQDL_2 was proven in [5]. Thus, step two is sound, and establishes that $X = x$ for $Y = G^X \bmod N$, $y = g^x \bmod p$. Furthermore, step five is sound, given the soundness of Schnorr signatures (see [22]) and their extension to composite moduli (see appendix B.) Thus, this step establishes that (a^2, b^2) is a valid encryption of 1, using public key Y and modulus N. Thus, $(a^2, b^2) = (Y^\beta, G^\beta)$. This implies that the plaintext must be a root r of unity, and that $(a, b) = (Y^\alpha r, G^\alpha)$, where $\beta \equiv_{\varphi(N)} 2\alpha$. The CA verifies (in step six) that b have Jacobi symbol 1. Therefore, since Y is a power of G (as established in step 2), a/r is a power of b. Since a has Jacobi symbol -1, so must r. In step 6, it is established that -1 has Jacobi symbol 1, and (obviously), the same holds for the value 1. Therefore, the plaintext r must be a non-trivial root of 1. As was outlined in the key extraction protocol for $\mathsf{Arc}_{DL \to RSA}$, knowledge of such a value allows straighforward factoring of N. Since knowledge of the proprietary discrete log private key x implies knowledge of the decryption key X (as established in step 2), we see that anybody with knowledge of the proprietary key can compute the private collateral key, which concludes the proof.

Security. The security of the RSA $\rightarrow$ DL and the RSA $\rightarrow$ RSA arc creation protocols is the same as in the Poupard-Stern fair encryption protocols [21]. Namely the proofs are zero-knowledge, and the ciphertext is with respect to the Paillier cryptosystem which is semantically secure under the hardness of distinguishing N-th residues modulo N^2 (see [20]). For the DL $\rightarrow$ RSA protocol, the proofs are zero-knowledge.

One needs to assume, however, that the key-translation protocol does not weaken the hardness of the discrete log problem. For this, we rely on a variant of the DDH assumption. Normally, this assumption is applied over a single group $\mathcal{G}$ of order q. It states that for generators μ_1 and μ_2 drawn uniformly at random, and exponents a, b drawn informly at random from $\mathbb{Z}_q$, it is computationally infeasible for a polynomial-time entity to distinguish between the two distributions $D_1 = \{\mu_1, \mu_2, \mu_1^a, \mu_2^a\}$ and $D_2 = \{\mu_1, \mu_2, \mu_1^a, \mu_2^b\}$.

We introduce a variant assumption that we call the *cross-group DDH assumption*. We consider two groups $\mathcal{G}_1$ and $\mathcal{G}_2$, where the order of $\mathcal{G}_1$ is q, and that of $\mathcal{G}_2$ is at least q. The distributions D_1 and D_2 are constructed exactly as above, except that μ_1 is a generator of $\mathcal{G}_1$ and μ_2 is a generator of $\mathcal{G}_2$. In other words, the cross-group DDH assumption as applied to $\mathcal{G}_1$ and $\mathcal{G}_2$ states that it is infeasible to test equality of discrete logs across groups. We apply this assumption in our paper to two groups for which the conventional DDH assumption is believed to be hard. The cross-group DDH assumption may be seen to arise in implicit form in earlier literature such as, e.g., [5], and seems a potentially important assumption for a wide range of protocols.

The ciphertext E is an ElGamal encryption of a non-trivial root r of unity, and such a r does not belong to the subgroup $\mathcal{G}$ spanned by G because it has Jacobi symbol -1. But the semantic security of ElGamal under the DDH assumption over $\mathcal{G}$ relates to plaintext in $\mathcal{G}$. However, one can easily notice that if ElGamal with plaintexts chosen in the kernel of the Jacobi symbol is semantically secure (which is equivalent to the DDH assumption in that kernel, which itself is believed to be true), then ElGamal with plaintexts having Jacobi symbol -1 is also semantically secure. Indeed, if an attacker is able to build two particular plaintexts m_0 and m_1 having Jacobi symbol -1, and to determine with non-negligible advantage if a challenge ciphertext (of either m_0 and m_1) is an encryption of m_0 or m_1, then he could also determine with non-negligible advantage if a challenge ciphertext c (of either m_0^2 and $m_0 m_1$) is an encryption of m_0^2 or $m_0 m_1$ (by division). Thus, since both m_0^2 and $m_0 m_1$ have Jacobi symbol $+1$, this would break the semantic security of ElGamal for plaintexts in the kernel of the Jacobi symbol. Note that the DDH problem for the kernel of the Jacobi symbol is believed to be hard when the modulus is a product of two safe primes (see [1]).

Unlinkability. We see that unlinkability follows from the fact that we use semantically secure encryption of the collateral private keys and the pointers to the collateral public keys and their associated CA; and that no information about *other* keys associated with a user is used or included in a signature using one particular public key.

Efficiency. All of the protocols require the inclusion of a ciphertext describing or pointing to the collateral public key. Additionally, $\mathsf{Arc}_{DL \rightarrow RSA}$ and $\mathsf{Arc}_{DL \rightarrow DL}$ require the inclusion of a generator G as described above. Assuming (probabilistically padded) RSA encryption is used when the proprietary key is an RSA key, and El Gamal encryption used when it is a discrete log key, the encryption of the "pointer" has size $|N|$ resp. $2|p|$.

The two arc establishment protocols that are directly based on Paillier encryption result in ciphertexts of size $2|N|$. The protocol using composite El Gamal alone results in cipher-

texts of that same size, while the protocol using both El Gamal encryption and Paillier encryption naturally results in ciphertexts of size $4|N|$.

Thus, for $|N| = |p| = 1024$ bits, the total certificate expansion is between $3|N| = 384$ and $4|N| + 2|p| = 768$ Bytes.