Theft-Protected Proprietary Certificates

Alexandra Boldyreva[1] and Markus Jakobsson[2]

[1] Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA
aboldyre@cs.ucsd.edu
http : //www − cse.ucsd.edu/users/aboldyre

[2] RSA Laboratories, 174 Middlesex Turnpike, Bedford MA 01730, USA
mjakobsson@rsasecurity.com
http : //www.markus − jakobsson.com

Abstract. The notion of *proprietary certificates* [8] was recently introduced in an attempt to discourage sharing of access rights to subscription-based resources. A proprietary certificate is a certificate on a public key – the so-called *proprietary* key – that contains some information related to another (so-called *collateral*) certificate and has the property that if the owner of the proprietary public key reveals the corresponding (so-called proprietary) secret key, then the collateral secret key (corresponding to the public key in the collateral certificate) is automatically released. Thus, if a service provider requires all users to use proprietary certificates linked with collateral certificates corresponding to resources the users always wish to keep private – such as access to 401(k) accounts, the user's criminal history, etc – then this will discourage the access rights sharing. However, the original solution for proprietary certificates overlooks the possibility of *accidental* sharing, namely, sharing caused by theft of the proprietary secret key which would lead to immediate loss of the collateral secret key, making wide-scale deployment of proprietary certificate approach unlikely. In this paper we discuss what steps can be taken towards making proprietary certificates approach more practical. While our solution preserves all the properties the original solution of [8] achieves, most importantly, protection against *intentional* rights sharing, it satisfies an additional property, namely, *theft protection.*

1 Introduction

1.1 Proprietary Certificates

One of the main goals of Digital Rights Management (DRM) is to protect digital content from illegal or inappropriate use. In various PKI applications, digital certificates can provide a partial solution to this problem. Namely, in order for a service provider to restrict access to appropriate users, it can verify each user's identity and authenticity of the user's public key by having the user present a valid digital certificate. However, this does not fully solve the problem. A registered user can share access rights to the resource (thus violating DRM policies) by lending his or her certificate and the corresponding secret key to another party.

The problem of preventing access rights sharing has been addressed in the literature. The high level idea of the existing solutions is to force a user who shares a secret key which gives access to some service to additionally share some other sensitive information, e.g. a credit card number. In most solutions (see, for example, [6, 15]) a service provider or a certification authority must be trusted since it learns this sensitive information.

A solution proposed recently by Jakobsson, Juels and Nguyen [8] attempts to solve the problem of access rights sharing without a trusted third party requirement. The authors introduce a *proprietary certificate*, which is a way of implementing digital certificate that discourages unwanted sharing of resources. A proprietary certificate may be used where a standard certificate may otherwise be employed but where the service provider wishes to discourage resource sharing. For example, it may be used for verifying and granting access rights to subscribers of a stock quote service while discouraging them from sharing their access with non-subscribers.

Subscribers are discouraged from sharing access rights with others by "linking" the secret key associated with the user's certificate (called the *proprietary* secret key and certificate) to a second secret key corresponding to another certificate (called the *collateral* secret key and certificate). The certificates which can serve as collateral ones are those which correspond to the accounts that are supposedly very important and long-lived such as bank accounts, 401(k) plans, health or criminal records, etc.

The link between the proprietary and collateral secret key guarantees that anybody with knowledge of the proprietary secret key can compute the corresponding collateral key. Thus, users who allow others access to the resource associated with the *proprietary* key are punished by automatically relinquishing control over the resource associated with the *collateral* key. If the collateral key grants access to the user's bank account or appending to his criminal record, this would clearly make sharing undesirable on a large scale.

More precisely, [8] considers a set of *users* and *certification authorities (CAs)* and proposes a certification protocol run between a user and CA in order to produce a proprietary certificate. We note, that CAs is in many cases can be functionally similar or identical to *service providers*, as the latter may often register users by pseudonyms or public keys – whether used internally or globally. For this reason, we will use the terms *service provider* and *certification authority* interchangingly onwards. At a high level, the approach of [8] is to include in proprietary certificates a ciphertext of a special form, where the secret key allowing the ciphertext to be decrypted is a proprietary secret key. Decrypting the ciphertext, in turn, results in the collateral secret key. In order not to require CAs to know proprietary and collateral secret keys of the users the solution of [8] uses fair encryption methods of Poupard and Stern [12]. We review fair encryption in Section 2.2 and discuss the solution of [8] and the properties it achieves in more detail in Section 3.

1.2 Vulnerabilities of Proprietary Certificates

While the original construction for proprietary certificates [8] achieves its stated goals, it overlooks the possible scenario in case of theft of the proprietary secret key that would lead to immediate loss of the collateral secret key. Hence, theft of the proprietary key grants the intruder full access rights to all resources associated with both the proprietary and corresponding collateral keys. In other words, their approach punishes not only *intentional* sharing, but also *accidental* sharing. This would, with a big likelihood, make large-scale deployment of such a scheme unlikely, given the threat of device loss and theft, burglaries, and computer viruses.

1.3 Our Goals

In this paper, we suggest measures which can help to overcome the weakness of the proprietary certificates approach and to make it more practical. We try to balance the requirements to punish intentional sharing with the desire to avoid penalizing accidental sharing, which is a seemingly contradictory problem. We propose a proprietary key certification protocol where the resulting proprietary certificate has an additional *theft protection* property. Theft-protected proprietary certificates meet all the properties of proprietary certificates and satisfy an additional theft-protection property. Namely, derivation of a collateral secret key from the proprietary secret key and the proprietary certificate is possible only after some predefined time delay. During this time delay, however, no information about a collateral secret key can be obtained, even by the party with knowledge of a proprietary secret key. Thus, the advantage is that the legitimate user has time to take measures in case of accidental exposure of a proprietary secret key.

Such approach would be useful for the settings where it is reasonable to assume that the collateral secret key is stored more securely than the proprietary secret key and that the owner of the device containing proprietary keys would know of a potential compromise of them. For example, if the proprietary keys reside on a palmtop computer or a cellular phone, loss of such device would indicate secret key compromise. Similarly, if the proprietary keys reside on a desktop computer, it would be prudent to assume key exposure if the office containing the computer is burglarized or if security software detects viruses, hacker activities or intrusion. For various less obvious techniques which help to detect secret key compromise see [9].

If the user reports theft of his key within this time period, the access keys for the proprietary and collateral services may be cancelled and re-newed, preventing both the accidental and intentional “share holder” from accessing either resource. Potentially the user can repeatedly share re-newed keys, however, various policies can be enacted to avoid this – a user who time after time cancels and re-keys an account may not get a renewed key after some time. This will satisfy both requirements, as it would provide security against theft and losses, while still discouraging sharing.

1.4 The Solutions

Technically speaking, the core of our solutions can be described as implementing time delays for the routine for deriving collateral secret keys from proprietary secret keys and certificates. We provide the solutions for time delays measured in real time or delays measured in CPU time. Our solution for the real time delay requires that the proprietary CAs support additional interaction with the parties regarding release of information related to collateral keys. In contrast, the CPU time delay solution does not require such communication, since the derivation of the collateral key can be done locally by any party with knowledge of the proprietary secret key and certificate. While CPU time is relative to processor speed, our solution withstands efforts to speed up the derivation of collateral keys by the use of parallelization of computation. Both of our solutions use fair encryption, and the CPU delay solution uses the notion of time-lock puzzles introduced by Rivest, Shamir and Wagner in [13]. We review the latter in Section 2.3 and describe our solutions in detail in Section 4.

In Section 5 we discuss various techniques of how service providers can help detecting secret keys compromise, and describe how additional features, such as user alerts can be implemented on top of our basic protocols, and used according to suitable policies.

1.5 Business Model

It is clear that subscription services benefit from the establishment of proprietary certificates to discourage their users from sharing access to the resources with others. Honest users, in turn, may also benefit from such an arrangement by a reduction of subscription fees and improved access, where the latter is due to the decreased access by non-registered users. Turning to the collateral accounts, there are two distinct situations. A first type of collateral account has the property of being *intrinsically* valuable to its owner, or where limited access rights are vital. Examples of this type is a 401(k) account or an account with append access to criminal records. These accounts will exist independently of the use of proprietary certificates. A second type of account is *artificially* made to become valuable. For this second type, one needs to provide an incentive to create and maintain accounts. We suggest the possibility of a class of service providers *whose sole business* is to support collateral accounts. Thus, such parties would certify users after having received a deposit or other security from them, and would either charge the user for the service (the interest, say), or receive periodical payments from the proprietary service providers. Note that users may establish *one* such "collateral account" and use it as collateral for several service providers.

2 Building Blocks

Here we outline the existing primitives and notions we will use in our work. Since our solutions support two main key types used in proprietary and collateral certificates we first recall the structure of RSA and discrete-log based keys. Next we review the notion of fair encryption [12] that is used in [8] and our solutions. Finally we outline time-lock puzzles introduced in [13], which we use for our CPU-time-delay solution.

2.1 Types of Keys

RSA keys Let k be the security parameter. In order to create an RSA type key pair a user picks at random two k-bit primes p, q and computes $N = pq$. He then picks a random number $e \in Z_N$ coprime to $\phi(N)$, where $\phi(\cdot)$ is a Euler's totient function. The public key of the user is (N, e). The user also computes d such that $ed \equiv 1 \bmod \phi(N)$. The secret key of the user is (N, d). Usually $k = 512$. These types of keys are used in the standard RSA encryption and signature schemes [11] and in many others schemes and protocols.

Discrete-log (DL) keys Let k be the security parameter. In order to use discrete log (DL)-based keys, a user picks at random a k-bit prime p and a prime q such that q divides $p-1$. He then picks a random generator g of the group of order q. He picks a random element x of Z_q and computes $y = g^x$. The public key of the user is (p, q, g, y) and the corresponding secret key is (p, q, g, x). Often all users use the same values p, q, g. Usually $k = 1024$. These types of keys are used in El Gamal encryption and signature schemes [7], Cramer-Shoup encryption scheme [4], Schnorr signature schemes [14], etc.

2.2 Fair Encryption

Following the definition given in [2], a *verifiable encryption* is a two-party protocol between a prover P and a verifier V who initially have access to some public key pk_1, some public value p and some binary relation R. At the end of the protocol the verifier obtains a ciphertext under pk_1 of some value x and accepts if the relation R between x and p holds and rejects otherwise. The properties of the protocol is that V can accept "invalid" x only with negligible probability and that V learns nothing about x.

A *fair encryption* is a verifiable encryption where the relation is true if x is a secret key sk_2 corresponding to the public key $pk_2 = p$. In other words, the prover convinces the verifier that a given ciphertext is a valid encryption of the secret key sk_2 corresponding to pk_2 which can be decrypted using sk_1, the secret key corresponding to pk_1 such that the verifier does not learn anything about sk_2. Poupard and Stern [12] give efficient solutions for the fair encryption of the RSA- and discrete-log-type secret keys using the Paillier encryption scheme [10]. As noted in [8], the protocols of [12] are applicable for the case when pk_1 is RSA-type key since the Paillier encryption (resp. decryption) can be performed under RSA public (resp. secret) keys (wlog we assume that Paillier public key G is equal $N + 1$, where N is a public modulus of pk_1). [8] provides the solution for fair encryption of both types of secret keys under the discrete-log-type public keys. Their solution for fair encryption of RSA type keys under the DL-type keys has a an additional requirement that a user's RSA modulus N be a product of two *safe* primes p, q, where $(p-1)/2$ and $(q-1)/2$ are both large primes.

Therefore, there exist solutions for fair encryption of both types of secret keys under the both types of public keys. All the protocols of [12, 8] can be made non-interactive, so in this paper we will assume so. We will use the notation $\mathcal{FE}_{pk_1}(sk_2)$ to denote the fair encryption of the secret key sk_2 under the public key pk_1.

2.3 Time-Lock Puzzles

Rivest et al. [13] provide a solution for the problem of encrypting a message in such a way so that no one can decrypt it until a pre-defined amount of time has passed. It might seem that the problem has a trivial solution, namely, one should encrypt the message using some symmetric encryption scheme using some not very long key. Then in order to decrypt this ciphertext one would need to do exhaustive key search which would take some time depending on the length of the key. As [13] notes, this solution is not satisfactory. First, a brute-force key search is parallelizable and second, the actual running time of the decryption process will depend on the order in which the keys are examined.

We now sketch the solution of [13]. Assume A wants to encrypt a message M with a time-lock puzzle for a period of T seconds. A picks at random two large primes p, q and computes $n = pq, \phi(n) = (p-1)(q-1)$. She then computes $t = TS$ where S is the number of squarings modulo n per second that can be performed by the potential decryptor. Then A picks a long random key K for some secure symmetric encryption scheme and encrypts M using K. Let us call the resulting ciphertext C_M. She then computes $C_K = K + a^{2^t} \bmod n$ for some random $a, 1 < a < n$. Since A knows $\phi(n)$, she can do this efficiently. The time-lock puzzle will contain (n, a, t, C_K, C_M). In order to extract M anybody would need to compute a^{2^t} and the only way to do this without knowing $\phi(n)$ is to perform t sequential squarings. The time delay ensured by this solution is not really absolute real time but some time period depending on the CPU power of the decryptor. We will refer to this as CPU time delay.

3 Proprietary Certificates

Let CA_1, (resp. CA_2) be the distinct certification authorities issuing the certificates for the proprietary (resp. collateral) services. Let C_1, (resp. C_2) be the proprietary (resp. collateral) certificates of some user. Assume that C_1, C_2 are publicly available. We now review the desirable properties of the system; for more details see [8].

- *Non-transferability*. Any user who learns secret key of C_1 would be able to compute the secret key of C_2, thus, reducing the likelihood of transferring proprietary certificates.

- *Cryptosystem agility*. Proprietary and collateral services can use different cryptosystems. For example, the secret key of C_1 can be RSA type and the secret key of C_2 can be discrete-log based key.
- *Locality*: CA_1 does not need to interact with CA_2 directly. However, the "light" version of interaction such as broadcast of information by CA_2 is necessary. See the discussion below.
- *Efficiency*: The certificate C_1 should not be substantially larger than a regular certificate of its type without proprietary properties.
- *Security*. Any party does not learn any information about the secret key of C_2. No party besides CA_1 learns what other certificates the user has. CA_1 learns only what public key and certificate the user uses to access collateral service.

The paper [8] shows how to extend the regular certificate to make it proprietary one, namely being linked to the collateral certificate. As it was suggested in [8], fair encryption can be used for the implementation of proprietary certificates. More precisely, the standard certification process of the public key of the user is modified as follows:

In order to certify the public key pk_1, the certification authority CA_1 (which acts as a proprietary one) asks the user to present the certificate of the another key pk_2 issued by CA_2 which he uses for some other service (to be considered as collateral) and the value $F = \mathcal{FE}_{pk_1}(sk_2)$ which is the fair encryption of his collateral secret key sk_2 under pk_1. If CA_1 agrees to use this certificate as collateral (if the potential loss of the collateral secret key would prevent the user from sharing his proprietary secret key), she then verifies validity of the collateral certificate by checking the signature of CA_2 and validity of the fair encryption. The properties of fair encryption ensure that CA_1 does not learn any information about the user's collateral secret key while being able to verify whether this ciphertext is valid. CA_1 also needs to be sure that pk_2 is still a valid key. It is assumed that CA_2 broadcasts the updates to the list of valid public keys. Thus CA_1 needs to check that pk_2 is still on that list. No direct interaction between CA_1 and CA_2 is required. If verification is successful, then CA_1 includes F and the encryption of pk_2 under pk_1 in the certificate in addition to standard information such as the user's identity information and pk_1. If the user shares sk_1 with another party, then that party can decrypt F and obtain sk_2. It is shown in [8] that this approach allows us to achieve the properties sketched above.

As we mentioned in the introduction, the weakness of the above approach comes from the fact that accidental exposure of a proprietary secret key due to theft or intrusion would immediately lead to a loss of the collateral key. Such scenario is possible since the proprietary keys are supposedly less valuable than collateral ones and, therefore, can be stored on less secure devices. Therefore, direct use of proprietary certificates would be risky since it imposes additional insecurity on the collateral secret key: no matter how well its storage is protected, its security can be violated through exposure of the less secure proprietary key.

As a result of this problem it is unlikely that the proprietary certificates approach be of wide practical use.

4 Making Proprietary Certificates Theft Protected

In this paper we discuss what steps can be taken towards making proprietary certificates approach more practical. At the first glance, the problem of key lending prevention and the problem of theft protection might seem contradictory. Indeed, the former requires the entity with possession of the proprietary secret key to be able to compute the collateral secret key while the latter would ask to prevent this possibility. However, we show that a compromise is possible.

While our approach is based on the proprietary certificates solution of [8] and preserves all the properties it achieves, our solution has one more additional property, namely, *theft protection*. The theft-protection property is a modified non-transferability property we discussed above. Namely, we require that in case of involuntary proprietary key exposure the user has time to detect the fact of theft and to contact proprietary and collateral service providers. During this time delay no entity, even the one with knowledge of the proprietary secret key should be able to derive the collateral secret key. After that delay (but not before), however, the entity with possession of the proprietary secret key should be able to obtain the collateral secret key as has been required before by the non-transferability property.

As we discussed in the introduction, we assume that the users are able to detect key theft within some period of time. The necessary time delay should be determined depending on the factors such as how fast the user can detect intrusion, contact service providers, etc.

We now provide our main solutions. We show how to implement the certification protocol run by a user and a proprietary CA in order to produce a theft-protected proprietary certificate. We prove that the resulting certificates meet the requirements of proprietary certificates and also have theft-protection property. Namely, we first show that even possessing the proprietary secret key no information about the collateral secret key can be obtained during some preset time delay and secondly we show how the collateral secret key can be derived after the delay.

The first of our solutions describes the implementation of CPU time delay, which does not require additional participation of the CA. Our second solution presents the realization of real time delay. In this case, however, additional involvement of the CA is required.

4.1 Implementing a CPU Delay

We use the idea of time-lock puzzles from [13], as outlined in Section 2.3, in order to implement a CPU delay for the link between the proprietary and collateral secret keys.

Let $\mathcal{U}$ denote the user that wants to certify the proprietary public key pk_1 with the proprietary certification authority CA_1. We assume that $\mathcal{U}$ holds proprietary and collateral public and secret key pairs $(pk_1, sk_1), (pk_2, sk_2)$ and the certificate on the collateral key C_2 signed by CA_2 that contains standard information such as U's identity info ID_U and the collateral public key pk_2.

We let $\mathcal{FE}$ denote the fair encryption algorithm and $\mathcal{SE}$ be some semantically-secure symmetric encryption algorithm with some appropriately chosen key length k. We may use a symmetric cipher such as AES with a 128-bit key in CBC mode [5, 1].

Let T be the desirable time delay in seconds, and let S be the approximate number of squarings required to unlock the puzzle, where all squarings are performed modulo some composite n, chosen by CA_1.

By combining time-locks and encryption under the proprietary public key, we obtain the desired functionality.

Certification protocol. In order to produce a theft-protected proprietary certificate C_1 on a public key pk_1, the following interactive protocol is executed by $\mathcal{U}$ and CA_1:

1. $\mathcal{U}$ computes $F = \mathcal{FE}_{pk_1}(sk_2)$ (see Section 2.2 for details). He then sends (ID_U, pk_1, F, C_2) to CA_1.
2. CA_1 verifies ID_U, C_2 and whether F is a valid fair encryption of the collateral secret key (we refer to [8] for a description of these steps). If it is incorrect, then CA_1 aborts; otherwise she continues as follows:
   (a) She picks two large random primes p, q and computes $n = pq$.
   (b) She picks a random k-bit string K and computes $E_F = \mathcal{SE}_K(F)$, where k is large enough such that exhaustive search done in polynomial time is not possible.
   (c) She computes values a, b as a function of pk_1. (We provide the details of how a, b are computed below for RSA and DL keys). Wlog we assume that $n > a$.
   (d) She computes $E_K = K + a^{2^t} \bmod n$, where $t = TS$.
   (e) Finally, she composes the certificate C_1 which contains $(ID_U, pk_1, E_F, E_K, n, t, b)$ and a valid signature on this data and returns C_1 to $\mathcal{U}$.
3. CA_1 sends $\phi(n)$ to $\mathcal{U}$ securely (encrypted under pk_1 using any secure encryption scheme).

We now specify how the values a, b above are computed.

Use of RSA keys. First, we will consider the case when $\mathcal{U}$ holds RSA-type proprietary keys. Assume his proprietary public key is (N, e) and the corresponding secret key is (N, e), see Section 2.1 for details. Then CA_1 picks some random number $a \in Z_N^*$ and computes $b = a^e \bmod N$.

Use of DL keys. Now, consider the case when $\mathcal{U}$ has discrete-log-type proprietary keys. Suppose his proprietary public key is (p, g, q, g^x) and his secret key is x, refer to Section 2.1. Then CA_1 picks some random $r \in Z_q$ and computes $b = g^r$ and $a = y^r = g^{rx} \bmod p$. If CA_1 holds discrete-log-type keys as well, then we can simplify this by making use of the CA's keys. If CA_1 has the public key (g, q, g^y) and the corresponding secret y, she can put $b = g^y$, which is a part of CA's public key and compute $a = (g^x)^y = g^{xy} \bmod p$.

Claims. We now show that the above protocol has the desired properties. First of all, note that due to the properties of the proof of correctness of fair encryption the CA does not get any information about sk_2. Next note that since the CA knows the factorization of n, she can compute a^{2^t} efficiently, namely, she can compute $s = 2^t \bmod \phi(n), E_K = K + a^s \bmod n$. Next, we claim that not knowing sk_1 it is not possible to compute any information about sk_2 due to semantic security of $\mathcal{FE}$. Hence, security property is satisfied.

Now suppose that some party P learned $\mathcal{U}$'s proprietary secret key sk_1. In any case the only way for P to compute sk_2 is to decrypt E_F in order to get F and to decrypt it using sk_1. P cannot do exhaustive key search for K within polynomial time because the key is long. Note that for any types of the $\mathcal{U}$'s keys P can compute a as a function of b and sk_1. For the RSA-type keys P computes $a = b^d \bmod n$. For DL-type keys P computes $a = b^x \bmod p$. As [13] shows, the only way for P to decrypt E_F is by computing a^{2^t} and then K. And since P does not know the factorization of n, he can do so only by performing sequential squarings which take time at least T. After that P can decrypt E_F, F and obtain sk_2. P can also compute pk_2 and find the corresponding collateral certificate C_2, since we assumed that all certificates are public. Thus, the theft-protection property is preserved. It is easy to see that the protocol satisfies the rest of the properties of theft-protected proprietary certificates.

$\mathcal{U}$ can verify that E_F, E_K are composed correctly using $\phi(n)$ as follows. He computes $s = 2^t \bmod \phi(n)$, $K = E_K - a^s \bmod n$, decrypts E_F and compares the result with F.

It remains to mention that it is not possible to pre-compute the value of a from b without the knowledge of the proprietary secret key.

4.2 Implementing a Real-Time Delay

Herein, we consider an approach in which a party has to interact with a CA in order to complete the derivation of the collateral secret key.

Certification protocol. As with regular proprietary certificates, during the process of certification of a public key, the user sends to the proprietary CA his proprietary public key, some proof of identity, and the collateral certificate. In addition, he sends a fair encryption F of the collateral secret key under the proprietary public key, one component of which is a proof of correct contents.

As before, the proprietary CA verifies the validity of all information, including the certificate associated with the collateral key, and the fair encryption. Now, however, she does *not* include F in the certificate, but rather stores it privately along with user's information in her database. We also assume that the CA and users agree on the use of a secure (unforgeable under chosen-message attack) signature scheme which uses keys of the same type as the one of the proprietary keys.

Derivation of collateral secret. If some party P obtains the user U's proprietary secret key sk_1, he would contact the CA which in turn would send P the random challenge value r. P computes a signature on r using sk_1 and sends it to the CA along with a public key pk_p for which P knows the secret key.

The CA searches for pk_1 in her database and verifies the validity of the signature. If it is correct, she waits the necessary time period and then returns the fair encryption F of the collateral key (as collected above), encrypted under pk_p. We stress that the CA needs to send F securely, or other parties could obtain it, allowing them to derive the collateral key of party U *immediately* after they obtain U's proprietary key (thus, the delay would only hold for the first request, in the worst case).

Upon receiving F, P (who knows sk_1) can decrypt F and obtain the collateral secret key.

Claims. The above shows that the party P with knowledge of the proprietary secret key can obtain the collateral secret key after the real time interval. It is clear that P cannot do it prior to this time, as he does not know F then. No party which does not know the proprietary secret key cannot obtain F since he cannot forge a valid signature. Thus, the theft protection property is preserved. It is easy to check that all the other properties are satisfied. We omit the details.

5 Alarm Techniques and Policies

Proprietary side alarms. It is clear that the trusted third party – the proprietary CA in the protocol of Section 4.2 – may sound an alarm once she receives a request for a fair encryption. By the proposed structure, it is clear that he will know what proprietary key has been compromised (in other words: the requests are not blinded with respect to what account they correspond to). We argue that it may not be in her best interests always to sound the alarm, though, as this provides cheaters with a "rescue mechanism". By making the alarm probabilistic, we can maintain the deterrence against sharing, while still allowing warnings to be generated when appropriate. We can use any policy, potentially made dependent on the status of the account in question, to determine when to alert users and collateral account holders.

Collateral side alarms. In the scenario in which the delay is governed by a CPU intensive task there would not be anybody to sound such an alarm, given that the party who is trying to retrieve the collateral key does not need to interact to do so. Let us therefore also consider the use of alarms on the collateral accounts as well as on the proprietary accounts. (These may be used for certificates with real time delay as well as those with CPU delay).

A collateral-side alarm can be achieved by requiring *two* keys to access a collateral account, one long and one short. Both have to be used to gain access to an account. The *long* key would be the one we have referred to as the collateral secret key. The *short* key, which may be as short as a few bits, does not have a public counterpart (and so, a guess cannot be verified). We refer to the short key as the *secret string*. This is not embedded in any certificates, whether proprietary or collateral, but merely used as a "trip wire". It is known by its owner, and by the CA corresponding to the associated account. When a user attempts to log in, he would not be given access permission of the collateral key is incorrect (whether the collateral string is or not). If both are right, he is given access. Otherwise, it is up to local policies whether to give access, and whether to sound the alarm. These, and other actions, may be probabilistic, and may be governed by arbitrary policies.

Acknowledgments

This work has greatly benefitted from discussions with Stanislav Jarecki. We also wish to thank Ari Juels and Chanathip (Meaw) Namprempre for their helpful feedback. Alexandra Boldyreva was supported in part by SDSC Graduate Student Diversity Fellowship, NSF Grant CCR-0098123 and NSF Grant ANR-0129617.

References

1. M. Bellare, A. Desai, E. Jokipii and P. Rogaway, "A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation," Proceedings *of the* 38th *Symposium on Foundations of Computer Science*, IEEE, 1997.
2. Jan Camenisch, Ivan Damgård, "Verifiable Encryption, Group Encryption, and Their Applications to Group Signatures and Signature Sharing Schemes," *Advances in Cryptology – ASIACRYPT '00*, LNCS Vol. 1976, T. Okamoto ed., Springer-Verlag, 2000.
3. Dario Catalano, Rosario Gennaro, Nick Howgrave-Graham and Phong Q. Nguyen, "Paillier's cryptosystem revisited," *ACM Conference on Computer and Communications Security* 2001.
4. R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack," *Advances in Cryptology – Crypto '98*, LNCS Vol. 1462, H. Krawczyk ed., Springer-Verlag, 1998.
5. "DES modes of operation," *National Institute of Standards and Technology, U.S. Department of Commerce*, 1980.
6. C. Dwork, J. Lotspiech and M. Naor, "Digital signets: Self-enforcing protection of digital information," Proceedings *of the* 28th *Annual Symposium on Theory of Computing*, ACM, 1996.
7. T. ElGamal, "A public key cryptosystem and signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol 31, 1985.

8. M. Jakobsson, A. Juels and P. Nguyen, "Proprietary Certificates," *Proceedings of the The Cryptographers' Track at the RSA Conference 2002*, LNCS Vol. 2271, Springer-Verlag, 2002.
9. M. Just and P. van Oorschot, "Addressing the problem of undetected signature key compromise," *NDSS*, 1999.
10. P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," *Advances in Cryptology – Eurocrypt '99*, LNCS Vol. 1592, J. Stern ed., Springer-Verlag, 1999.
11. "PKCS-1," RSA Labs, `http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/`.
12. G. Poupard and J. Stern, "Fair encryption of RSA keys," *Advances in Cryptology – Eurocrypt '00*, LNCS Vol. 1807, B. Preneel ed., Springer-Verlag, 2000
13. R. Rivest, A. Shamir and D. Wagner, "Time-lock puzzles and timed-release crypto," *LCS technical memo MIT/LCS/TR-684, February 1996.*
14. Claus P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology, 4:161–174, 1991.*
15. S. Stubblebine, P. Syverson and D. Goldschlag, "Unlinkable serial transactions: protocols and applications," *TISSEC 2(4): 354-389, 1999.*