ASYNCHRONOUS LARGE-SCALE CERTIFICATION BASED ON CERTIFICATE VERIFICATION TREES

Josep Domingo-Ferrer, Marc Alba and Francesc Sebé
Universitat Rovira i Virgili
Dept. of Computer Engineering and Mathematics
Autovia de Salou s/n
E-43006 Tarragona, Catalonia, Spain
{jdomingo,malba,fsebe}@etse.urv.es

Abstract Good public-key infrastructures (PKIs) are essential to make electronic commerce secure. Quite recently, certificate verification trees (CVTs) have been introduced as a tool for implementation of large-scale certification authorities (CAs). In most aspects, the CVT approach outperforms previous approaches like X.509 and certificate revocation lists, SDSI/SPKI, certificate revocation trees, etc. However, there is a trade-off between manageability for the CA and response time for the user: CVT-based certification as initially proposed is synchronous, *i.e.* certificates are only issued and revoked at the end of a CVT update period (typically once a day). Assuming that the user is represented by a smart card, we present here solutions that preserve all advantages of CVTs while relaxing the aforementioned synchronization requirement. If short-validity certificates are used, implicit revocation provided by the proposed solutions completely eliminates the need for the signature verifier to check any revocation information (CRLs, CRTs, etc.).

Keywords: Large-scale public key infrastructures, Certification authorities, Certificate verification trees, Smart cards, Implicit revocation.

1. INTRODUCTION

The design of efficient large-scale public-key infrastructures (PKIs) for electronic commerce is a hot topic of research in these days. Two important bottlenecks are certificate revocation (signalled as a bottleneck as early as seven years ago, [1]) and the design of manageable large-scale certification authorities.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35413-2_36

R. Steinmetz et al. (eds.), *Communications and Multimedia Security Issues of the New Century*

Most currently proposed CAs assume that digital certificates consist of a c-statement (the statement to be cerfified, *e.g.* an individual's name, a public key and an expiration date) along with the CA's signature which validates the c-statement. One drawback of such an assumption is that the authorization given by the CA's signature may need to be revoked at some future time. Several solutions to deal with revocation have been proposed:

- Certificate revocation lists (CRLs) are the standard solution for revocation that is being deployed with X.509 certificates.

- To reduce the huge size of large-scale CRLs, a special kind of Merkle trees [5] called certificate revocation trees or CRTs is used to store revocation information in [6, 4, 7].

- Another solution is to assume that no revocation is needed because certificates are good until their expiration date [2, 9]. Using short-validity certificates is one way to make such an assumption realistic [8].

- In [3], CRLs and CRTs are eliminated as separate data structures for revocation information. Actually, both revocation information and the certificate directory are stored by the CA in the same Merkle tree, called certificate verification tree (CVT).

The solution based on CVTs is the most scalable one, but it also has the drawback of tying certificate issuance and revocation to CVT updates. For efficiency reasons, CVT updates are only carried out once in a period (*e.g.* on a daily basis), which means that all certificate requests or revocation requests during the period will not be processed before the period's end. Assuming that the user is represented by a smart card, we describe in this paper solutions that maintain all advantages of CVTs while allowing for asynchronous certificate renewal and revocation. If short-validity certificates are used, implicit revocation provided by the proposed solutions completely eliminates the need for the signature verifier to check any revocation information (CRLs, CRTs, etc.).

In Section 2, we recall the advantages of CVTs with respect to previous proposals. In Section 3, we describe our proposed solution. Section 4 analyzes the strong points of our proposal in terms of efficiency and security. A variant of the basic scheme designed to minimize communication is described in Section 5. Conclusions are summarized in Section 6.

2. CERTIFICATE VERIFICATION TREES

In the CVT approach described in [3], the CA constructs a Merkle B-tree: each leaf is a c-statement plus the hash of that c-statement (obtained using a collision-free hash function). The hash values of siblings in the B-tree are hashed together to get a hash value for their parents node; this procedure iterates until the root node of the B-tree is obtained. Then the CA signs the hash value of the root node, called RV, together with some additional information such as date and time. The *cert-path* for a c-statement is the path from the leaf containing the c-statement to the root, along with the hash values needed to verify that path (which include the hash values of siblings of nodes along that path). When a user requests a certificate, the CA supplies additionally the cert-path of the c-statement, plus the signature on RV.

As new certificates are issued by the CA, these are incorporated to the Merkle B-tree; tree update is performed on a regular basis, *e.g.* once a day. If the B-tree is a 2-3 tree, each certificate update requires only $O(\log n)$ work for the directory, where n is the total number of certificates. For each batch of certificates that is incorporated to the CVT, only one signature is computed on RV; the remaining computations are hashes, which using Rivest's figure are about 10000 times faster than signatures.

CVTs allow the CA key to be changed following compromise or just as a security measure, for example to increase the key length. The CA just has to generate a new key pair, broadcast its new public key and sign the current RV with the new key. New RV's arising from tree updates will be signed with the new private key as well. Note that in previous schemes where certificates are signed individually, a CA key change requires to revoke all currently valid certificates, issue new certificates and forward those to their owners.

The ability to deal with historical queries is another strong point of CVTs. Certificate usage may be ephemeral (*e.g.* session authentication) or persistent (*e.g.* signature on a real-estate purchase). For persistent usage a requirement of non-repudiation appears: the user should be able to prove in some years from now that her public key was valid when using the certificate (*e.g.* when signing the purchase order). For historical queries to be possible with CVTs, the CA should store the roots of the CVTs (one root per update period). For any contract needing persistent certificates, the cert-paths of the necessary c-statements should be stored with the contract, along with the signature on RV. In case of CA key change, all stored root values should be signed with the new key. Note that, in previous schemes with individually signed certificates, allowing

historical queries for persistent certificates requires the CA to store all certificates ever issued, along with complete CRLs for every update period; also a CA key change implies resigning all ever issued certificates and old CRLs!

Proving certificate non-existence is also an important feature of CVTs. CRLs and CRTs do not provide evidence of the existence of non-revoked certificates. A CVT can comfortably store all certificates ever issued by a CA: using a CVT of height 30 (which yields cert-paths of length 30) is enough to store more than 10^9 certificates (which is more than the existing number of VISA credit cards). Thus, certificate forgery is difficult to hide.

Last but not least, CVTs give the possibility of verifier caching for efficiency. Verifiers checking many signatures every day (like vendors or routers using certificates for session authentication) may reduce communication with the CVT directory and computation by caching the top part of the CVT. This top part together with the CA signature on RV need only be retrieved and verified once per update period (*e.g.* once a day). Further, if a CVT has depth d and the top d_1 levels of it are cached, a signature verification will only require $d - d_1$ hashes.

3. CVT-BASED ASYNCHRONOUS CERTIFICATION

As mentioned in the introduction, the weak point of CVTs as used in [3] is that certificate issuance and revocation are synchronous: they are performed only at the start of each CVT update period. With daily updates at midnight, this means that a certificate request placed at 1:00 AM would have to wait 23 hours before being serviced. The same holds true for a revocation request. When comparing with the traditional solution of individually signed certificates, it becomes apparent that the improved manageability of the certificate directory described in Section 2 is darkened by delay in reacting to issuance and revocation requests.

We present in this section a CVT-based scheme whereby batches of certificates can be requested ahead of time without the user having to store the corresponding private keys (which would increase the chances of key compromise). In practice, this allows the user to use a new certificate as soon as the current one has expired, which is functionally equivalent to asynchronous certification —no synchronization with CVT updates is needed—. When mentioning short-validity certificates in what follows, it will be assumed that such certificates do not survive more than one CVT update period; if necessary, adjust the validity of the last certificate in a CVT period for its expiration to occur at the end of the period. As

an example, if the CVT is updated once a day, then a one-hour validity would be appropriate for a short-validity certificate. It is assumed that the signer is represented by a smart card SC or another tamper-resistant device. The scheme can be described by three protocols: set-up, signature and implicit revocation upon loss of signer's smart card. In the description of those protocols, we distinguish communication between SC and the CA from communication between SC and the CVT directory (for certificate download).

In the set-up stage, a batch of public-private key pairs is computed ahead by the signer and stored by the CA in the CVT. Key pairs are assumed to be valid in consecutive future time intervals, *i.e.* if the first key pair in the batch is valid in time interval t, then the i-th key pair is constructed to be valid during the $t + i - 1$-th time interval. The aim of the solution proposed here is to make intervals of certificate validity completely independent from CVT update periods. Let m be the number of key pairs in the batch (m can be arbitrarily large).

Protocol 1 (Set-up)

1 *The signer's smart card SC generates a key k for a symmetric block cipher* (e.g. *DES, AES).*

2 *For* $i = 1$ *to* m*:*

   (a) *SC generates a public-private key pair* (pk_i, sk_i).

   (b) *SC encrypts* sk_i *under k to get* $E_k(sk_i)$.

   (c) *SC sends* $(pk_i, E_k(sk_i))$ *to the CA.*

   (d) *SC deletes* pk_i, sk_i *and* $E_k(sk_i)$ *from its memory.*

3 *CA stores the encrypted private keys* $E_k(sk_i)$ *received from SC.*

4 *In the next update of the general CVT, CA adds all received* pk_i*'s to the CVT.*

We assume that Protocol 1 is run frequently enough so that one never runs out of key pairs for the next time intervals; the larger m, the less frequently the protocol needs to be run. Under this assumption, the signer can use the following protocol to sign at a time interval t for which a valid public key exists in the CVT:

Protocol 2 (Signature at time interval t)

1 If the user's smart card SC stores a private key sk_t valid for time interval t, then SC does:

(a) If the certificate for pk_t is not a short-validity one and this is the first signature computed during the current CVT update period, get from the CVT directory the current cert-path for the pk_t certificate. The cert-path may have changed as a result of the CVT update. (Short-validity certificates are not supposed to survive more than one CVT update period, so there is no need to refresh cert-paths for them).

(b) Sign using sk_t (which involves appending the certificate and the cert-path of pk_t to the signature).

2 Otherwise SC does:

(a) Delete from its memory the last private key sk_j stored, if any.

(b) Get $E_k(sk_t)$ from the CA.

(c) Get the certificate and the cert-path of pk_t from the CVT directory.

(d) Decrypt $E_k(sk_t)$ to get sk_t.

(e) Sign using sk_t.

Under the above scheme, SC stores no private key except the current one, but can use a new certificate as soon as the current one has expired (provided that the new one was requested ahead in the past). If the SC is stolen or lost, the following revocation protocol is invoked by the user:

Protocol 3 (Implicit revocation)

1 The user informs the CA on the loss of SC.

2 The CA stops serving encrypted private keys $E_k(sk_i)$ to SC.

Note that, if Protocol 3 is run at time interval t, it implicitly revokes all certificates in the CVT which were requested by SC for time intervals $t' > t$. The reason is that the CA will not provide $E_k(sk_{t'})$ for the SC to sign at time interval t'. The current certificate is not revoked by Protocol 3. To eliminate the need for explicit revocation of the current certificate, short-validity certificates can be used. In this way, an intruder will have little time to tamper with SC in order to have it sign under the private key SC is currently storing.

4. PERFORMANCE ANALYSIS

The advantages of the proposed asynchronous CVT-based scheme are outlined in what follows. Note that the scheme maintains all of the advantages of CVTs listed in Section 2.

4.1. EFFICIENCY

The following are strong points regarding efficiency:

- *Asynchronous certification.* Requesting certificates from the CA ahead of time allows the *SC* to use a new certificate as soon as the current certificate expires. From a functional point of view, this is equivalent to asynchronous certification.

- *Reduced storage.* The user can request a batch of certificates, but her smart card only needs to store the current key pair and the key of a symmetric block cipher.

- *Implicit revocation.* As mentioned above, in the event of smart card loss or compromise, Protocol 3 provides implicit revocation for all certificates requested by that smart card for time intervals following the current one. Explicit revocation of the current certificate can be made unnecessary if short-validity certificates are used. Further, note that implicit revocation does *not* require a CVT update (unlike the explicit revocation used in [3]).

4.1.1 Implicit vs explicit revocation. As pointed out in the previous paragraph, implicit revocation combined with short-validity certificates can completely eliminate the need to keep explicit revocation information; no CRLs, CRTs or CVT updates are needed.

Implicit revocation is safer for the signer and especially convenient for the verifier. Under the traditional paradigm, explicit revocation is used, which is implemented by labelling the certificate of a public key as revoked. As pointed out in [8], explicit revocation is consistent with the traditional certificate guarantee that "this certificate is good until the expiration date, unless, of course, you hear that it has been revoked". Thus, explicit revocation forces the CA to distribute CRLs or update CVTs and, even worse, it also forces the signature verifier to check those CRLs or CVTs before accepting a signature. If a certificate is to be revoked before starting its validity interval, *implicit revocation is a much better alternative which prevents the private key corresponding to a revoked certificate from ever being used to sign*; in this way, the need for verifiers to check revocation information is reduced.

4.1.2 Communication requirements. A seemingly weak point of Protocol 2 is that communication between SC and the CA is required each time a certificate expires. However, this holds for competing approaches as well. Indeed, both for individually signed certificates and for the CVT solution [3], the user requests a new certificate from the CA when the current one is about to expire. Requesting in advance a batch of certificates in those approaches is not practical because it would force the user to find secure storage for the whole corresponding batch of private keys.

On the other hand, if the amount and frequency of CA-user communication required by short-validity certificates is deemed too high to be affordable, then one can resort to longer-lived certificates. In that case, explicit revocation of the current certificate is a sensible option: it can be either based on CRLs or on a CVT update (the leaf containing the current certificate is labelled as "revoked"). CVT updates do not require separate revocation information (such as CRLs) to be maintained, but they have the drawback that revocation will not become effective before the next update period.

4.2. SECURITY

If the user's smart card SC is lost or stolen and Protocol 3 is run by the user at time interval t, there is a guarantee for the user that SC will become useless from time interval $t+1$ onwards. Regarding time interval t, the following situations are possible:

- At the moment of being lost/stolen, SC has not yet downloaded sk_t (because no signature has been performed since time interval t started). In this case, SC may contain sk_j for some $j < t$ but this private key has expired. No further action by the CA is required, since SC is useless to the intruder.
- SC has already downloaded sk_t, but the corresponding certificate is a short-validity one (*e.g.* a one-day certificate). In this case, it is unlikely that the intruder can hack SC to sign with sk_t before certificate expiration. Thus, probably no further action needs to be taken by the CA.
- SC has already downloaded sk_t and the corresponding certificate is a long-lived one. In this case, the CA should use explicit revocation (CRLs or CVT update) to revoke the certificate corresponding to sk_t.

In any of the above cases, it can be seen that SC contains only one private key (the one that was most recently used); therefore, compromise

of the SC does not jeopardize past key pairs. The other key contained by SC is the key k used to encrypt private keys in the current batch. Tampering with SC to get k is useless, because Protocol 3 ensures that no encrypted private keys will be served by the CA in the future.

The use of a secret sharing scheme can help increasing dependability and reducing the trust requirements of the scheme proposed in this paper. For example, assume an n_1-out-of-n_2 threshold scheme [10] is used in Protocols 1 and 2 as follows:

- In Protocol 1, the user's smart card computes n_2 shares of $E_k(sk_i)$ using the threshold scheme. Then each share is sent to a different CA.
- In Protocol 2, the user's smart card polls all n_2 CAs to recover $E_k(sk_t)$; getting the shares of $n_1 \leq n_2$ CAs is enough for recovery to succeed.

The increase in communication and computation caused by the secret sharing scheme may be justified by the following benefits:

Dependability Up to $n_2 - n_1$ CAs polled in Protocol 2 may be unreachable and the encrypted private key can still be recovered. If a single CA is used as described in Section 3, then unreachability of the CA causes Protocol 2 to fail (the new private key cannot be obtained).

Increased security Secret sharing schemes (and more specifically threshold schemes) exist which offer unconditional security that a collusion of less than n_1 CAs have no chances at all of recovering $E_k(sk_i)$. This adds to the computational security offered by the symmetric block cipher that sk_i cannot be recovered from $E_k(sk_i)$ without knowledge of k. Remark that symmetric encryption cannot be eliminated unless we assume that collusions of n_1 or more CAs will not happen (if the shared secret was sk_i instead of $E_k(sk_i)$, a collusion of n_1 CAs could recover the user's private key).

5. A VARIANT FOR REDUCED COMMUNICATION

Smart cards are equipped with low-bandwidth I/O interfaces, so reducing communication in on-line transactions may be an important goal. As pointed out in Paragraph 4.1.2, explicit revocation of the current certificate may be a solution if the amount of communication needed to use short-validity certificates is too high. Explicit revocation allows

longer-lived certificates to be used, which require less frequent communication. However, the communication cost is transferred from the signer to the verifier, who will be forced to check revocation information (CRLs, CRTs, etc.) before accepting a signature.

Rather than trying to reduce the frequency of communication, an alternative strategy that does not transfer any cost to the verifier is to *use short-validity certificates, while minimizing the size of messages exchanged between the CA and SC.* This is possible by modifying Protocols 1, 2 and 3 as follows:

Protocol 4 (Set-up)

1 For $i = 1$ *to* m*:*

- *(a) SC generates a key* k_i *for a symmetric block cipher.*
- *(b) SC generates a public-private key pair* (pk_i, sk_i).
- *(c) SC encrypts* sk_i *under* k_i *to get* $E_{k_i}(sk_i)$.
- *(d) SC sends* (k_i, pk_i) *to the CA.*
- *(e) SC stores* $E_{k_i}(sk_i)$.
- *(f) SC deletes* pk_i, sk_i *and* k_i *from its memory.*

2 CA stores the keys k_i *received from SC.*

3 In the next update of the general CVT, CA adds all received pk_i*'s to the CVT.*

Protocol 5 (Signature at time interval t)

1 If the user's smart card SC stores a private key sk_t *valid for time interval* t*, then SC signs using* sk_t.

2 Otherwise SC does:

- *(a) Delete from its memory the last private key* sk_j *stored, if any.*
- *(b) Get* k_t *from the CA.*
- *(c) Get the (short-validity) certificate and the cert-path of* pk_t *from the CVT directory.*
- *(d) Decrypt* $E_{k_t}(sk_t)$ *to get* sk_t.
- *(e) Sign using* sk_t.

Protocol 6 (Implicit revocation)

1 The user informs the CA on the loss of SC.

2 The CA stops serving keys k_t to SC.

In Protocol 5, the communication between the CA and *SC* is reduced to the CA sending a symmetric block cipher key (typically 128 bits) each time a new certificate is to be used. In the basic proposal (Protocol 2), an encrypted private key was sent, which could take a few kilobits. Similarly, in Protocol 4 less bits are sent from *SC* to CA than in Protocol 1. However, unlike in the latter protocol, the size m of the batch of encrypted private keys that can be generated ahead in a single execution of Protocol 4 is limited by the amount of available storage in *SC*. The more storage, the larger m can be and the less frequently is set-up execution needed.

6. CONCLUSION

Certificate verification trees are a very convenient data structure for managing large-scale public key directories. As initially proposed, their only major drawback is that certificate issuance and revocation must be synchronized with the start of a CVT update period. To sort out this problem, we have proposed a scheme where certificates can be requested by the user and added ahead of time to the CVT without decreasing security; no one but the user's smart card device knows private keys, but only one private key must be safely stored by the user's smart card at a time. Upon expiration of the current certificate, the new one can be obtained from the CVT; this renewal is asynchronous in that it does not depend on the CVT update periods (provided that enough certificates have been added in advance to the CVT). In the event of loss or compromise of the user's smart card, all certificates in the CVT linked to that card (except the current one) are implicitly revoked because the CA will not supply the encrypted private keys corresponding to those certificates.

References

[1] S. Berkovits, S. Chokhani, J. A. Furlong, J. A. Geiter and J. C. Guild. *Public Key Infrastructure Study: Final Report.* The Mitre Corporation, 1994.

[2] C. M. Ellison. *SPKI Certificate Documentation*, 1998. `http://www.clark.net/pub/cme/html/spki.html`

[3] I. Gassko, P. S. Gemmell and P. MacKenzie. Efficient and fresh certification. In *Public Key Cryptography'2000*, pages 342-353, 2000. Springer-Verlag LNCS 1751.

[4] P. Kocher. *A quick introduction to certificate revocation trees (CRTs)*, 2000. `http://www.valicert.com/technology`

[5] R. Merkle. A certified digital signature. In *Advances in Cryptology - Crypto'89*, pages 218-238, 1990. Springer-Verlag, LNCS 435.

[6] S. Micali. Efficient certificate revocation. In *RSA Data Security Conference.* San Francisco CA, January 1997.

[7] M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proceedings of 7th Usenix Security Symposium.* San Antonio TX, January 1998.

[8] R. L. Rivest. Can we eliminate certificate revocation lists? In *Financial Cryptography'98*, pages 178-193, 1998. Springer-Verlag, LNCS 1465.

[9] R. L. Rivest and B. Lampson. *SDSI-A Simple Distributed Security Infrastructure*, 2000. `http://theory.lcs.mit.edu/~cis/sdsi.html`

[10] D. Stinson. *Cryptography: Theory and Practice.* CRC Press, 1995.