On the Synergy Between Certificate Verification Trees and PayTree-like Micropayments

Josep Domingo-Ferrer

Universitat Rovira i Virgili
Dept. of Computer Engineering and Mathematics
Av. Països Catalans 26
E-43007 Tarragona, Catalonia
jdomingo@etse.urv.es

Abstract. A substantial number of micropayment schemes in the literature are based on distributing the cost of a digital signature over several successive micropayments (*e.g.* Payword). Thus, a stable relationship between user and merchant is assumed: the micropayments validated by the same signature must take place between the same user and merchant. This stability is ill-suited for surfing on the web, a situation in which successive micropayments are made to different merchants. Thus coin-based micropayments, in which successive micropayments can be unrelated to one another, are far more interesting. One practical coin-based micropayment system is PayTree, which is amazingly similar to PKIs based on certificate verification trees (CVTs). We propose in this paper a synthesis of a CVT-based PKI with a PayTree-like micropayment system. The proposed system achieves a threefold reduction of the cost of public key certification and coin processing through: 1) sharing certificates by two applications (public key and multi-coin certification); 2) including certificates in the CVT on a batch basis (costs decrease as the batch size increases); 3) accelerating certificate (and thus public key and coin) verification through the verifier/merchant caching feature of the tree structure used. Furthermore, certificate and coin verification costs also become smaller as the number of users increases.
Keywords: PKI and eCommerce, Certificate Directories, Protocols, Coin-Based Payments, Micropayments, PayTree, Certificate Verification Trees.

1 Introduction

Micropayments are electronic payments of low value and they are called to playing a major role in the expansion of electronic commerce: example applications are phone call payments, access to non-free web pages, pay-per-view TV, etc. Standard electronic payment systems (like CyberCash [5], e-cash [8] , *i*KP [2], SET [18]) for low-value payments suffer from too high transaction costs as compared to the amount of payments. The cost of transactions is kept high due to complex cryptographic protocols like digital signatures used for achieving a certain security level. However, micropayments do not need as much security as speed and simplicity (in terms of computing). For that reason and with few

S.K. Katsikas et al. (Eds.): EuroPKI 2004, LNCS 3093, pp. 180–190, 2004.

exceptions [16,13,3], micropayment proposals try to replace the use of digital signatures with faster operations, like hash functions (whose computation is, according to figures by Rivest [17], about 10000 times faster than RSA signature computation).

Micropayment systems based on hash functions include NetCard [1], μ-iKP [10], PayWord [17], MicroMint [17], PayTree [11] and spending programs [6]. With the exception of MicroMint and PayTree, the above schemes use hash chains which represent chains of micropayments authenticated by a single signature on the first link of the hash chain. In this way, the cost of the digital signature is split over several successive micropayments. The price paid is that stable user-merchant relationships must be assumed: successive links of the hash chain can only be verified by the merchant who verified the signature on the first link. This stability assumption is shared by other micropayment schemes not using hash functions [16,13].

In MicroMint, the bank generates microcoins by computing collisions of a hash function. The appeal of this coin-based system is that successive micropayments can be quite unrelated. Thus one user may send one micropayment to merchant 1, the next micropayment to a different merchant 2, and so on. The problem with MicroMint is that its use of hash collisions requires very specific computational assumptions on the user, the merchant and the bank (see [19]).

PayTree uses a completely different approach to provide unrelated micropayments. It is a system based on Merkle's authentication tree scheme [14] and uses the following structure: a tree whose leaf nodes are labeled by secret random values, whose internal nodes are labeled by the hash value of the nodes' successors and whose root is signed. Because of the tree structure, PayTree is more flexible than PayWord-like systems in that it allows payments to different merchants to be made using different parts of the tree. Thus, multiple merchants can share the cost of a public-key signature.

While the need for micropayment systems assuming stable user-merchant relationships has recently been debated (why not use subscription or flat rate instead?, see [20]), the usefulness of unrelated micropayments for Internet surfing remains unquestioned.

1.1 Our Contribution

We propose in this paper to build a PayTree-like coin-based micropayment system upon an existing PKI based on certificate verification trees [9,7]. Due to the similarity between PayTree and CVTs, their combination in a single system results in a threefold reduction of the cost of public key certification and coin processing:

- The same certificates are used for two applications (public key and multi-coin certification);
- Certificates are included in the CVT in batches;
- Certificate (and thus public key and coin) verification benefits from verifier/merchant caching made possible by the tree structure used.

It is the low cost of coin processing what makes low coin denominations and thus micropayments possible with the proposed system.

In Section 2, we recall the advantages of CVTs with respect to other public-key certificate management systems. In Section 3, we describe our proposed solution. Section 4 analyzes the strong points of our proposal in terms of efficiency and security. Conclusions are summarized in Section 5.

2 Certificate Verification Trees

Like PayTree, CVTs are based on Merkle trees. Let a c-statement be a statement containing the name of an individual, her public key and an expiration date. In the CVT approach described in [9], a certification authority 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 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 of a public key, CA additionally supplies the cert-path of the c-statement, plus the signature on RV.

As new certificates are issued by 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. 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 (like X.509, [4]), 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, 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. In previous schemes with individually signed certificates (*e.g.* X.509), allowing historical queries for persistent certificates requires CA to store all certificates ever issued, along with complete Certificate Revocation Lists (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 (Certificate Revocation Trees, [12,15]) 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.

The weak point of CVTs as used in [9] 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 is darkened by the delay in reacting to issuance and revocation requests. Assuming that the user is represented by a smart card, we presented in [7] a solution that maintains all advantages of CVTs while allowing for asynchronous certificate renewal and revocation. The idea of the scheme is that batches of public-key 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). The user is represented by a smart card which performs all user functions. We will use that solution as the basis of the micropayment scheme in this paper.

3 Tree Micropayments in an Asynchronous CVT Public-Key Directory

The basic idea of the combination between micropayments and public-key certification is to certify coins together with public keys. From now on, the CA maintaining the CVT will also take the role of bank (we assume for simplicity that all users and merchants use the same bank). A certificate will contain a public key and several coins. Just as each certified public key corresponds to a *private* key needed to sign, each coin corresponds to a *secret* key needed to spend

the coin. The secret key of a coin is first revealed when the coin is spent during payment.

It is assumed that the user is *always* represented by a smart card SC or another tamper-resistant device. The scheme can be described by four protocols: certificate construction, certificate refreshment, payment and implicit revocation upon loss of user's smart card. In the description of those protocols, we distinguish communication between SC and CA from communication between SC and the CVT directory (for certificate download).

In the request protocol, a batch of m certificates is computed ahead of time by the user and stored by CA in the CVT. All certificates considered in the rest of this paper will be short-validity ones; more specifically, the i-th certificate in the batch contains a public key and n_i coins and is constructed to be valid during the i-th next CVT update period. Having established that successive certificates in a batch are assumed to be valid in consecutive CVT update periods, we will omit validity information in what follows in order to keep the notation simpler. Storing a certificate in the CVT by CA is an implicit CA signature on that certificate.

Protocol 1 (Certificate construction)
*We assume that, in some set-up stage, the user's smart card SC has generated and installed a key k for a symmetric block cipher (*e.g. *DES, AES). All communications in this protocol are assumed to be authenticated (via shared-key or public-key encryption).*

1. *For $i = 1$ to m:*
   (a) *SC generates a public-private key pair (pk_i, sk_i).*
   (b) *SC generates n_i coin secret keys, $ck_1, \cdots, ck_{n_i}$ and chooses n_i coin denominations $d_1, \cdots, d_{n_i}$.*
   (c) *SC computes $C_j = H(ck_j||d_j)||d_j$ for $j = 1, \cdots, n_i$, where H is a one-way collision-free hash function and $||$ is the concatenation operator.*
   (d) *SC encrypts $sk_i, ck_1, \cdots, ck_{n_i}$ under k to obtain $E_k(sk_i, ck_1, \cdots, ck_{n_i})$.*
2. *SC sends to CA the certificate $(pk_i, C_1, \cdots, C_{n_i})$ and the secret information $E_k(sk_i, ck_1, \cdots, ck_{n_i})$, for $i = 1$ to m.*
3. *SC deletes from its memory all items related to the batch of certificates (those items computed in Steps 1a through 1d).*
4. *In the next update of the general CVT, B adds*

$$pk_i||C_1||\cdots||C_{n_i}||status||info$$

to the CVT, for $i = 1$ to m. (Actually all coins sent by all users during the last update period are added to the CVT.) The field status contains as many bits as coins; initially, all bits are 0; in the next CVT update after C_j is spent, the j-th bit of status will be set to 1. The field $info$ contains the rest of information usual in a certificate (owner,issuer,validity, etc.).

In the above protocol, the choice of n_i depends on how many coins the user plans to spend during the validity period of the certificate. On the other hand,

the batch size m depends on the storage capacity of the smart card SC. The objective is for the user not to run out of valid certificates for the successive CVT validity periods; therefore, the larger m, the less frequently needs Protocol 1 to be run. Keeping this in mind, the user can go through the following protocol to obtain a current certificate at any time t:

Protocol 2 (Certificate refreshment at time t)

1. *If the user's smart card SC contains a certificate valid for time t then exit the protocol. (Note that, since short-validity certificates are used which are not supposed to survive more than one CVT update period, one never needs to refresh cert-paths for current certificates.)*
2. *Otherwise SC does:*
   (a) *If the secret information of an expired certificate corresponding to a previous time t' is still in the card, delete from SC's memory the private key $sk_{t'}$ stored and the coin secret keys related to the t'-th certificate, if any.*
   (b) *Get from CA $E_k(sk_t, ck_1, \cdots, ck_{n_t})$. Decrypt this value to get $sk_t, ck_1, \cdots, ck_{n_t}$. (Before supplying the encrypted values to SC, CA in his role as a bank deducts from the account of SC's owner the value $\sum_{j=1}^{n_i} d_j$ of coins contained in the t-th certificate. CA reimburses SC's owner for the value of any unspent and deducted coins contained in expired certificates.)*
   (c) *Get the certificate*

$$pk_t||C_1||\cdots||C_{n_t}||status||info$$

   and its cert-path from the CVT directory.

Protocol 2 will be run each time a user wants to sign or pay. After completing Protocol 2, a user can sign using sk_t (which involves appending the certificate and the cert-path of pk_t to the signature). In order to pay, the user can spend the coins contained in the certificate for time t using the protocol below.

Protocol 3 (Payment)
Assume that the user has obtained a current certificate using Protocol 2 and wants to spend the j-th coin C_j contained in that certificate.

1. *The user sends to the merchant the whole certificate containing the coin*

$$pk_t||C_1||\cdots||C_j||\cdots||C_{n_t}||status||info$$

   as well as the cert-path of the certificate and the secret key ck_j corresponding to C_j.
2. *The merchant checks that C_j is included as an unspent coin in the certificate and that the certificate is included in the CVT directory (this latter check is equivalent to checking that C_j was certified by CA). This step can be simplified and communication with the CVT minimized as explained below.*

3. *The merchant checks the correctness of the secret key, by testing*

$$H(ck_j||d_j)||d_j \stackrel{?}{=} C_j$$

4. *If the checks in the previous step are OK, the merchant accepts a payment amounting d_j from the user.*
5. *Immediately before the next CVT update, the merchant sends all accepted coins to CA to redeem them. In his role as a bank, CA credits the merchant for the amount of those coins among those submitted by the merchant which had not yet been spent. In the next CVT update, all coins redeemed since the last CVT update will have their status changed from "unspent" to "spent" (this will result in a 1-bit change in the status field of the CVT leaves — certificates— containing those coins).*

The verifier/merchant can reduce her computation and communication needs at Step 2 of Protocol 3 (verification of a coin) by taking advantage of the verifier caching feature described in Section 2 for certificates in CVTs. If the merchant receives many payments every CVT update period (*e.g.* every day), she can cache the top part of the CVT. This top part together with CA's signature on the root value RV need only be retrieved and verified once per update period. Thus, if a CVT has depth d and the top d_1 levels of it are cached, a coin verification will only require $d - d_1$ hashes (verification of the $d - d_1$ lowest hashes of the cert-path of the certificate containing the coin) and no communication with the CVT.

If the SC is stolen or lost, the following revocation protocol is invoked by the user:

Protocol 4 (Revocation)

1. *The user informs CA on the loss of SC using some form of personal authentication (biometrical, handwritten signature via fax, etc.).*
2. *CA stops serving the encrypted information of certificates in future instances of Protocol 2.*
3. *CA marks the status of coins in future certificates submitted by SC as "spent". Those unspent coins marked as spent will be reimbursed by CA to the account of SC's owner.*

Protocol 4 does not revoke public keys nor coins till the next update period. However, if update periods are short enough, the thief will not be able to tamper with SC to have the card sign or run the payment protocol (Protocol 3) within the current CVT update period. For that assumption to be realistic, the card owner's identification must be moderately secure (*e.g.* a biometric procedure or a four-digit PIN with a limited number of typing attempts).

4 Performance Analysis

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

- The certification authority CA can easily change his key;
- Historical certificates (and thus historical coins) can be dealt with;
- Certificate (and thus coin) non-existence can be proven.

We next examine further strong points specifically related to coins.

4.1 Efficiency

The following are strong points regarding efficiency:

- *Shared cost of coin certification.* One public key and several coins are packed in a certificate and certificates are included in the CVT *in batches.* This reduces the per coin communication and authentication cost: SC sends multi-coin certificates in batches and the cert-paths are also returned by the CVT on a batch basis (Protocol 1). In addition, a single digital signature is computed by CA for all multi-coin certificates received within a CVT update period; thus, the cost of the digital signature is spread among all received coins and public keys to be certified.
- *Merchant caching for coin verification.* As explained in Section 3, a verifier/merchant verifying many public keys or receiving many coins (not necessarily from the same user) can save a lot of computation and verification by caching the top levels of the CVT during each update period. Thus the cost of running the payment protocol (Protocol 3) is also reduced by the use of CVTs.
- *Asynchronous coin certification.* Requesting certificates ahead of time allows SC to use as many public keys/coins as needed, regardless from the frequency of CVT updates (assuming that enough coins have been packed into the successive certificates).
- *Public status of coins.* The feature that all coins, whether unspent or spent, are visible in the CVT directory allows the status of a coin to be readily determined.

The fact that the cost of constructing and verifying certificates is shared by a batch of public keys and coins keeps the processing cost per coin and public key low. Beyond the batch effect, the cost is further reduced because the cost of the whole process is shared by two applications: public-key certification and micropayment. In this way, coins can have a low denomination because the cost of processing them is kept low.

4.2 Security

For a security discussion on public keys in asynchronous CVTs, see [7]. We concentrate in this section on the security for micropayments:

- *Coin counterfeiting.* Only coins included in the CVT are acceptable currency. For a user to be able to spend a coin this coin must have been included in the CVT packed in a certificate submitted by an *identified* smart card SC;

besides, the smart card owner must be backed by enough money in her account to recover the secret key of the coin (Step 2b of Protocol 2). Thus, there is no room for false coins.

- *Double spending.* The secret key corresponding to a coin must be supplied by a smart card SC for the coin to be acceptable. If the hash function is collision-free one-way, the secret key cannot be deduced from the certificate posted in the CVT. It remains to be shown that a coin cannot be spent twice (by the same user or by a merchant having received that coin legitimately). Clearly, a spent coin will be publicly marked as spent in the next CVT update. The problem is thus confined to double spending during the current CVT update period (*e.g.* during the current day). This type of double spending cannot be prevented but it can be tracked. Note that the CVT manager (CA) knows who ran Protocol 1 for the certificate containing a particular coin (the communication in Protocol 1 was an authenticated one). Thus:
  - When the bank is confronted to two redemption requests for the same coin, the two requestors and the user having requested the inclusion of that coin in the CVT are suspect (they all know the secret key of the coin).
  - Those three parties should explain what the coin flow was. If none of them confesses herself guilty, then the bank deducts the value of the double-spent coin from the account of each of them. In this way, the expected gain for double spenders is zero. Innocent punished parties will refrain in the future from accepting transactions from parties they had disputes with, which will add ostracism to zero-gain for real double-spenders.
- *Revocation of stolen coins.* If a smart card SC is lost or stolen and Protocol 4 is run by the owner of SC, there is a guarantee for the user that SC and the coins it stores will become useless from the next CVT update period onwards. As noted above, if update periods are short enough, the thief will not be able to tamper with SC to have the card run the payment protocol (Protocol 3). For that assumption to be realistic, the card owner authentication mechanism implemented in the card must be secure enough so that bypassing it takes longer than what remains of the current CVT update period.

5 Conclusion

This paper highlights some connections between public-key certificate directories and coin-based payment systems. Certificate verification trees are a very convenient data structure for managing large-scale public key directories and PayTree is a very convenient and flexible micropayment system. In this paper, we have shown how to exploit the CVT-PayTree similarity to integrate public-key certification and coin-based micropayments in a synergetic way that further reduces the processing cost per public key and per coin. In our proposal, certificates in the CVT pack one public key with a bunch of coins spendable during the

validity period of the certificate. To facilitate revocation, the validity period of certificates is short and matches the duration of a CVT update period.

The cost of public key certification and coin processing is reduced in three ways: 1) certificate sharing by two applications (public key and multi-coin certification); 2) certificate inclusion in the CVT on a batch basis; 3) lighter certificate (and thus public key and coin) verification thanks to the verifier/merchant caching feature of the tree structure used. The resulting scheme yields coin processing costs which become smaller as the batch size and the number of redeemed coins increase.

The smart card used for generating coins must be tamper-resistant, since it stores the secret information needed to sign and pay during the current CVT update period; at the same time, the owner identification mechanism built into the card should not be bypassable within a CVT update period.

Acknowledgment

This work was partly supported by the Spanish Ministry of Science and Technology and the European FEDER Fund under contract no. TIC2001-0633-C03-01 "STREAMOBILE".

References

1. R. Anderson, C. Manifavas and C. Sutherland, "NetCard - A practical electronic cash system", 1995. Available from author: `Ross.Anderson@cl.cam.ac.uk`
2. M. Bellare, J. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik and M. Waidner, "*i*KP - A family of secure electronic payments protocols", in *First USENIX Workshop on Electronic Commerce*, New York, July 1995.
3. M. Blaze, J. Ioannidis and A. D. Keromytis, "Offline micropayments without trusted hardware", in *Financial Cryptography'2001*, pp. 21-40, 2002. Springer-Verlag, LNCS 2339.
4. CCITT (Consultative Committee on International Telegraphy and Telephony), *Recommendation X.509: The Directory-Authentication Framework*, 1988.
5. CyberCash Inc., `http://www.cybercash.com`
6. J. Domingo-Ferrer and J. Herrera-Joancomartí, "Spending programs: A tool for flexible micropayments", in *Information Security-ISW'99*, pp. 1-13, 1999. Springer-Verlag, LNCS 1729.
7. J. Domingo-Ferrer, M. Alba and F. Sebé. Asynchronous large-scale certification based on certificate verification trees. In *Communications and Multimedia Security*, eds. R. Steinmetz, J. Dittmann and M. Steinebach, Norwell MA: Kluwer Academic Publishers, pp. 185-196, 2001.
8. e-cash, `http://www.digicash.com`
9. I. Gassko, P. S. Gemmell and P. MacKenzie, "Efficient and fresh certification", in *Public Key Cryptography'2000*, pp. 342-353, 2000. Springer-Verlag, LNCS 1751.
10. R. Hauser, M. Steiner and M. Waidner, "Micro-payments based on *i*KP", IBM Research Report 2791, presented also at SECURICOM'96. `http://www.zurich.ibm.com/Technology/Security/publications/1996/HSW96.ps.gz`

11. C. Jutla and M. Yung, "PayTree: "amortized-signature" for flexible micropayments", in *Proc. of the 2nd USENIX Workshop on Electronic Commerce*, pp. 213-221, 1996.
12. P. Kocher, "On certificate revocation and validation", in *Financial Cryptography'98*, pp. 172-177, 1998. Springer-Verlag, LNCS 1465.
13. M. S. Manasse, "The Millicent protocols for electronic commerce", in *Proc. of the 1st USENIX Workshop on Electronic Commerce*, July 1995.
14. R. Merkle, "A certified digital signature", in *Advances in Cryptology - Crypto'89*, pp. 218-238, 1990. Springer-Verlag, LNCS 435.
15. M. Naor and K. Nissim, "Certificate revocation and certificate update", in *Proceedings of 7th USENIX Security Symposium*, San Antonio TX, January 1998.
16. T. Poutanen, H. Hinton and M. Stumm, "NetCents: A lightweight protocol for secure micropayments", in *Proceedings of the 3rd USENIX Workshop on Electronic Commerce*, September 1998.
17. R.L. Rivest and A. Shamir, "PayWord and MicroMint: Two simple micropayment schemes", in *Security Protocols Workshop 1996*, pp. 69-87, 1997. Springer-Verlag, LNCS 1189.
18. Secure Electronic Transactions.
`http://www.mastercard.com/set/set.htm`
19. N. van Someren, "The practical problems of implementing MicroMint", in *Financial Cryptography'2001*, pp. 41-50, 2002. Springer-Verlag, LNCS 2339.
20. N. van Someren, A. Odlyzko, R. Rivest, T. Jones and D. Goldie-Scot, "Does anyone really need micropayments?", in *Financial Cryptography'2003*, pp. 69-76, 2003. Springer-Verlag, LNCS 2742.