PayTree:
"Amortized-Signature" for Flexible MicroPayments

Charanjit S. Jutla* **Moti Yung†**

Abstract

We present the idea of *PayTree*, a method to amortize the work of a single signature production and verification among numerous micropayments in the model where payments are made from a payer to a set of merchants (under various trust assumptions regarding these merchants).

The PayTree scheme is simple yet flexible. Its main feature is that it can support arbitrary (dynamically determined) number of payees while using a single signature (unlike PayWord). It is therefore suitable for supporting users who "shop around". The scheme is easily extendible dynamically (without the use of further signatures) and has a reasonable computational penalty. It can be viewed as a "divisible coin" mechanism as well.

Remark: This work appeared in its original form in the 3rd Usenix Workshop on Electronic Commerce, Sep 1998 (this is a modified version). The work did not get well indexed and the basic result has been replicated in a few publications since then. Publication dissemination has some value!

*IBM T.J. Watson Research Center

†Bankers Trust Electronic Commerce/ CertCo

1 Introduction

A payment system needs to be efficient, so that the cost of operation of a transaction is favorable compared to the monetary value of the transaction itself. This is the logic behind various systems and designs.

A computational bottleneck in many systems is the public-key operations like "signature" [4, 17] which is essential in assuring the validity of payment in cash and micro-payment systems.

To solve the problem, a suggestion that spreads a signature value over many cryptographic values derived by more efficient cryptographic one-way (hash) function [12, 15] was suggested by Shamir and Rivest recently [16], and independently by various other researchers (Pedersen [14] and Anderson et al. [1].) The mechanism, known as PayWord, exploits Lamport's idea [8] (implemented as S/key). Namely, to sign a target value which is derived by many repeated applications of a one-way function, and opening the target value one step at a time in the chain of function applications; each time the opening looks like "inverting" the one-way function.

While being useful as a micropayment mechanism between two entities (a user and a merchant), PayWord does not provide a general purpose spending mechanism to be used by a user in a multi merchant system (namely, a user who

shops around in an electronic shopping center). In such an environment one prefers a signature which is made public and can be spent in many ways (especially when merchants are relatively honest but want to assure fast or cheap receipt of valid payment). ayWord In this paper we suggest "PayTree", a general purpose mechanism that amortizes a public-key signature scheme against spending of many coins, paid to either a single or to many merchants. The mechanism employs "authentication tree" techniques, it has preprocessing similar to PayWord, and when used in a single merchant (payee) environment, it has similar cost of payment (when amortized over many micropayment– which is the typical envisioned application for the mechanism). The mechanism enables to split the signature on micro-coins among merchants (and split the coin signed), so it can be called an "amortized-signature" mechanism.

PayTree has similar security guarantees as P (related model of trust and entity relationships, with some crucial changes based on multi payees as we will note), while providing better flexibility and multi payees. PayTree exploits Merkle [9] authentication tree and it also has an extendible "spending potential" using a "renewal capability" as in [10] (whereas PayWord is limited in this respect). The mechanism can be viewed as a "divisible coin" as well.

The mechanism can enhance systems which employ digital signature as a basic tool for amortization of signatures in payments and other "fixed" or "almost fixed" used-once authorization token schemes.

Remark added after we presented the work: We note that the mechanism is presented as all payments have the same value, and each tree has a pre-defined total value associated with it. But paytree can be slightly modified to implement trees with multiple denominations, unlimited payment potential or divisible coins. It can also be used as a module of other payment systems.

1.1 The model

Let us briefly review the "PayWord" model of Shamir and Rivest which we adopt and extend.

We assume that a user gets a signature scheme authorized (certified) by an authority (a broker) to issue "commitment to payment", that is it can allocate a budget for billing and then spend the budget in an incremental way. Alternatively the broker may sign strings representing monetary value directly.

Bills are respected by the broker and actual money transfer from the payer to the payee is performed. A bill spent has to be both "valid" and "uniquely spent".

The payee recognizes the certification and the signature, and has the means to recognize the "validity" of a bill it gets. The idea is to increment the bill as payments are transferred in a fast online fashion; whereas the broker is in charge of detecting "double spending", and it respects the first one and rejects the second payment with the same coin. More details of the model and trust relationships are in [16].

1.1.1 Assumptions on Merchants

We classify three models regarding payees (merchants):

- The single payee model.
- The multiple non-colluding payees model.
- The multiple potentially-colluding payees model.

Here we assume that based on a single commitment (posted in public, say) various merchants or a single merchant can get paid. The single merchant model is exactly as PayWord (i.e. the signature is associated with and is given to it), while, in general, PayTree extends PayWord's functionality. With multi payee setting, we have to make sure that we operate in an environment where merchants do not collude to present "double spending", or (especially in the case of big payments) that we have other mechanisms to cope with it (and this is doable as will be explained in the sequel). In the multi-payee setting signature are published and are not associated with a specific payee. The payment is associated with a receiver. Either a payment is given to a non-colluding merchant or is strongly associated with a (potentially misbehaving) merchant ID.

2 PayTree: the mechanism

2.1 The Basic PayTree

Let h be a one way function $h : \{0,1\}^n \rightarrow \{0,1\}^n$. Let
g_k be one way functions $g_k : \{0,1\}^{nk} \rightarrow \{0,1\}^n$. (Both functions can be based on fast cryptographic hash functions like SHA1 or MD5 (MD5 was suggested originally, nowadays SHA1, SHA256, and the coming SHA3 are viable alternatives)). A k-ary *PayTree* is the following labeled k-ary tree structure $\{N, r, \delta, L, h, g_k, F, R\}$:

- N is the set of nodes of the tree
- r is the root of the tree
- $F \subseteq N$ is the set of leaf nodes
- δ is the successor function $\delta : (N-F) \times [0..k-1] \rightarrow N$
- L is the labeling of the nodes of the tree $L : N \rightarrow \{0,1\}^n$
- h and g_k are one-way functions as above
- R is the secret labeling of the leaves $R : N \rightarrow \{0,1\}^n$. Thus the leaves have two labels.

In addition, the following holds for the PayTree:

1. For every node $s \in F$: $L(s) = h(R(s))$
2. For every node $s \in N - F$: $L(s) = g_k(L(\delta(s,0)), L(\delta(s,1)), ...L(\delta(s,k-1)))$

2.2 Signature

A party I wishing to pay using the PayTree, generates such a PayTree T and keeps the random numbers (R) labeling the leaves secret. I signs the label of the root r of the tree. Let this signature be called $S(T)$. In case, the protocol supports different height trees, the height of the tree is also included in the information signed. We assume that the trees are balanced, i.e. each leaf is at the same depth.

In fact, building on the same scenario as PayWord, the signature can be performed by a broker (or by a signature scheme certified by it). The broker later will collect the tree back and will detect double spending. The user "withdraw money or credit" from the broker by getting a signature on the tree; it then spends the tree leaves, paying to merchants. The merchants will give the leaves (coins) back to the broker who reconstructs the tree and checks for double spending. The tree is good for a limited time and the coins have to be redeemed within this period. Within a period,the broker keeps record of payment transfer together with the redeemed coins.

We can use any public key signature, RSA, DSS, etc. [17, 11].

2.3 Payment:

With each tree T, I maintains a list of merchants M, and for each merchant in M, it maintains a list of leaves F_i (where i is the index of the merchant in M) whose R values have already been disclosed to merchant i. Also, I maintains a list F_{all} which is the union of all the F_i.

Let the nodes of the tree be numbered in inorder fashion (with the first leaf numbered one). Thus the root is numbered k^h, and the last leaf is numbered $k^{h+1} - 1$ (where h is the height of the tree, and k is the arity of the tree).

To pay party m, I discloses to m the signature $S(T)$, and the label of the root r (these two and even the whole L can actually be made public knowledge). I checks if m is in the list M. If so, let i be the index of m in M. Otherwise, insert m in M, and let i be the index. In the latter case, create a new empty list F_i as well.

Next, I picks the smallest numbered leaf $f \in F$ not in F_{all}. Let t be the least common ancestor of $F_i \cup \{f\}$ (if F_i is empty, let t be the root). Let p be the path from t to f: $p_0 = t$, p_1, $...p_j = f$. Now, $L(p_0)$ has already been disclosed to m. Moreover, the merchant has already verified that $L(p_0)$ is indeed a valid label. Thus, to prove to m that $R(f)$ is the correct R-label of f, I discloses to m the secret $R(f)$ as the R-label of f, as well as the labels L of all nodes $q_{l+1,l'}$ (l' can range from 0 to $k-1$), such that $q_{l+1,l'}$ is a successor of some p_l, where $0 < l < j$, and $q_{l+1,l'}$ is not in the path p.

Disclosure of the R value of a leaf amounts to transfer of one unit of monetary value.

I updates both F_i and F_{all}.

2.4 Verification by merchant

The first time, the merchant m receives a PayTree T, it checks that the signature is a valid signature. Let i be its index in the list M maintained by I. By induction, assume that the R labels of all leaves in F_i have already been verified by m to match the signature of the T. In the process it has also verified the labels L of all nodes on all paths from leaves in F_i to r. In particular, it has verified the L-label of the least common ancestor t of F_i and f (the leaf received) Thus, to verify that the received R-value is the correct R-label of f, m validates the L label of f to be $h(R(f))$. To this end, it verifies that $L(p_0) = g_k(L(\delta(p_0, 0)), L(\delta(p_0, 1)), ...L(\delta(p_0, k-1)))$. One of the successors of p_0 is in the path p, and its L label is obtained recursively, whereas the other L labels have been received from I.

3 Complexity of PayTree

The cost of generating and operating a PayTree involves various factors:

1. Generating random numbers for the leaves. Assume that generating one random number costs C_r. In the PayTree, the R values can be generated using a pseudorandom generator, with a random seed. Thus, we can assume that $C_r = C_h$ (see next item).

2. Computing the labels L of the tree. Assume that computing either h or g_k costs C_h. Actually, computing g_k may be a linear function of k. However, for small k the discussion below still holds.

3. Signing the root label. Assume that signing costs C_s.

4. Verifying the signature. Assume that this costs C_e.

5. Finding least common ancestors during each payment. For a balanced tree, this operation is negligible in cost

6. Verifying the labels L during each payment

For sake of simplicity, we assume that the tree is binary. If the height of the tree is h, then the cost of generating the tree is $C_s + C_r + 2^h C_h + 2^{h+1} C_h$. If there is just one merchant, who receives all the 2^h payments, then the total number of verification steps (h computations) performed by this merchant is : 2^{h+1}. However, in the worst case, a single payment step may take $(h+1)C_h$ time. Thus, if each payment is made to a different merchant, this is the cost per payment for verification (in addition to C_e). However, if there are fewer than 2^h merchants, a single merchant will require at most $h(2^{p+1} - 1) - 1.44 * p2^p$ one-way computations per 2^{p+1} payments. Thus, for example, if $p + 1 = h - 1$, the amortized complexity per payment for a single merchant is at worst $0.28 * hC_h$ plus some lower order terms. However, this worst case occurs, when the payments to this merchant are interspersed with payments to other merchants in a particular manner. If the payments to a merchant are contiguous, the performance is more in the line of a small constant (i.e. independent of h) amortized one-way computations per payment.

To contrast with a different binary tree structure, let's consider a completely skewed binary tree. In this binary tree there is one long path of length 2^h, and from each internal node of this path, there is another child node which is a leaf (that is in F as per section 2.1). Again if there is just one merchant then the total number of computations required for all the 2^h payments is as in the balanced tree. However, if each payment is made to a different merchant, then on the average each merchant would require 2^h computations per payment.

4 Variations on PayTrees

Various additions and refinements are possible. We introduce a few of them.

4.1 Non-binary PayTrees

It is possible to use a larger degree trees rather than binary. It is also possible to change the degree of the tree in different levels, e.g., have a balanced binary tree where each end point connects to 20 branches at its last level, say. This last variation of degree-changing tree seems useful as it enables the "tagging" of leaves.

It is also possible to use a non-tree structures and sub-structures like in one-time signature dags (direct acyclic graphs) [2]. For simplicity, we do not include this option here.

4.2 PayTrees with Receiver Designation

Now we assume the case (trust model) where merchants may collude. This is an important extension where large payments are involved where the advantage of PayTree is obvious.

The basic idea is to "tag" a payment with the "name" of the merchant so that the payment is non-transferable.

Let us assume that merchants names are 10-bit long. Now we can use the degree-changing tree just mentioned. In the leaf level there are 20 applications of one-way function which are treated as 10 pairs. Each pair (a window) can sign a bit

by opening the preimage of the one-way function of the first value (for 0) or the second value (for 1). This is the Diffie-Lamport one-time signature scheme (see, e.g. [10]). A payment includes opening of the bits corresponding to the name of the merchants and the path in the tree to the root; this path is not to be reused. Now, in order to transfer a payment to another merchant the merchants have to forge a new one-time signature. There are other ways to "tag" a payee's name into a PayTree.

To implement such one-time signature efficiently one can use the ideas of Winternitz reported e.g. in [9]. Let us explain the mechanism. Essentially a row of PayWords is implemented (e.g., a five element row, each element can be opened as a digit from 0 to 9), and an extra check PayWord, which can be opened as 50 different values (a chain of size which is the sum of the chains of the rows). The rows are a register. We can go forward in the opening of the PayWord in each position, each prefix designating a different digit (so we can open in one of ten ways). We can reverse the one-way function at most 50 times on the rows. If in total we opened k function applications $k = \sum k_i$ where k_i is the value of the digit of position $i = 1, ..5$, The extra check PayWord is opened $50 - k$ times. This implies that changing any position in the register is impossible (the total length of opened PayWord chains is 50). Assuming the merchants are represented as a 5-digit number (10^5 merchants or "hashed names" of merchants), we have designed a mechanism that enables the opening of a leaf in a merchant-customized fashion. At this point if double spending to two different merchants happens, it is assumed that the payer is responsible for that faulty behavior. Note that the cost of such a mechanism is quite reasonable for a large merchant population, when the one-way function is really cheap.

Now, under the intractability of inverting the function, double spending detected by the broker actually implies wrong-doing of the payer, since the merchants cannot forge the signature scheme.

4.3 PayTrees with PayWords

To improve the performance, in cases where the payment to a merchant is interspersed with payments to other merchants, it is advantageous to have PayWord as leaves of the PayTree. If we implement the previous idea of receiver designation, we can add an initial value in addition to the "register" based on the row of initial values. The PayWord allow payments to be made to the same merchant in a further incremental fashion. However, a right balance needs to be struck as to the height of the PayWord and that of the PayTree.

4.4 PayTrees with multiple denominations

We first describe PayWord with multiple denominations, as this makes the exposition simpler. To achieve a multiple (k different) denomination payment mechanism (for a single merchant), one needs to build k PayWord using a one-way function h, and then their roots collapsed to a single value using g_k. This single value can then be signed and made public knowledge. It is previously decided among the three parties (bank, payer and merchant), as to what the denominations of the k different threads(PayWord) are. To pay a unit value in the ith denomination, the payer just discloses the next preimage in the ith PayWord.

In a PayTree, we could either have multiple denomination PayWord per leaf, or a single denomination PayWord per leaf (but different denomina-

tions for different leaves). The advantage of the latter is that, we can distribute the secret information to more merchants, for the same size of secret information. The disadvantage of the latter approach is that the merchant will have to verify more paths in the PayTree.

The latter approach can be made more advantageous by allowing the denomination of a leaf (and hence its PayWord) to be determined at the payment time. This is achieved by having at each leaf a set of windows (which allows the payer to disclose the correct denomination at the payment time, by opening the appropriate windows). The random numbers required for these windows can be pseudorandom, and the payer need not store all the pseudorandom numbers. With this approach the use of the information in the PayTree is more efficient.

4.5 PayTrees with unlimited payment potential

The PayWord at each leaf f can have the special property, that the mth pre-image of label L of f is g_k of k pairs of images of h, where k is the no. of bits in the range of h. Using the idea in [10] this allows for the disclosure of a new PayWord root of this type after $m-1$ payments. This process can continue indefinitely. Thus, after every $m-1$ payments, the merchant has to verify using a g_k computation. The "renewal" of a new PayTree authorized via an old PayTree can be put on a bulletin board, enabling further usage of the tree in a multi merchant environment.

This enables a flexible amortization of a public key signature.

4.6 PayTree as a divisible coin mechanism

One can view PayTree as a divisible coin into linkable sub-coins [13] (which built a number theoretic tree structure, rather than our more amorphic (any one-way function based) and (thus) efficient construction). One can spend merged sub-coins by paying with a subtree node which has a value which is the sum of its leaves. Another way is to view every opening along a path from the root as opening extra value (same as in PayWord).

5 What has been achieved

We claim a relatively simple yet strong mechanism:

1. A basic "amortized signature" as a micropayment mechanism for billing in the single or non-colluding merchant model, and with measures against such collusion when assumed. The broker can off-line get the coins (within a designated time limit) and transfer money between accounts. The users cannot cheat unless they can invert one-way hash functions (the actual proof relies on the hash function being an ideal "random-oracle like" function).
2. A way to combine the basic "amortized signature" with further refined amortization via the PayWord mechanism. Also, a way to designate various values to various parts of the mechanism.
3. An extendible mechanism to keep on amortizing a signature as far as we want without further use of a public key.
4. A way to implement a divisible coin.

The mechanism can be embedded in a transaction system to pay bills, to pay small amounts

of money (micropayments), and bigger amounts if measures against payees' collusions are implemented. An example that can use PayTree is a preparatory mechanism to handle "giving change" [3], where small residual values are expected. In fact, the

PayTree can modularly enhance existing mechanisms (e.g. [19, 18, 7]). For example, NetBill [18] involves the server in each transaction and signatures are used heavily, so PayTree can relax this potential burden and provide a lighter weight option. Netcash [7] is a design for payment that employs the trusted server on-line as well, but using the Kerberos model. PayTree can help as a divisible coin in such an environment. Millicent [5], on the other hand, uses symmetric keys and hash functions which makes it very "light weight." However, it relies on "trusting the merchants" while defending against customers. Our models, in contrast, either do not trust or only partially trust the merchants, yet we need to employ a signature scheme.

We remark that assuming that signature generation (e.g. RSA) is about 10^4 times slower than a one-way function/hash application (based on Message Digest like hash), a tree of size 1024 makes sense, or 512 assuming extra PayWord or receiver identifier tags are added to leaves.

We comment that the properties outlined above can be proved formally, assuming an ideal hash function (which acts as a random oracle) and one-way functions. Any ability to double spend, claim more money than actually was gotten, framing of payers, etc. implies an easy inversion of the assumed hard function.

References

[1] R. Anderson, H. Manifavas, and C. Sutherland, "A practical electronic cash system".

[2] D. Bleichenbacher and U. Maurer, "Direct acyclic graphs, one way functions and digital signatures", Proceedings of Crypto '94, pp. 75-82.

[3] E. Brickell, P. Gemmell, D. Kravitz, "Trustee based Tracing Extensions to Anonymous Cash and the Making of Anonymous Change," Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1995, pp. 457-466

[4] W. Diffie and M. E. Hellman, "New Directions in Cryptography," IEEE Trans. Info. Theory IT-22, Nov. 1976, pp. 644-654

[5] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro, "The Millicent Protocol for Inexpensive Electronic Commerce" The 4-th Int. WWW conference, 1995.

[6] S. Goldwasser, S. Micali and R. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," SIAM Journal of Computing 17(2), April 1988, pp. 281-308

[7] G. Medvinsky and B. C. Neuman, "Netcash: A design for practical electronic currency on the internet," The First ACM Conference on Computer and Communications Security, Nov. 1993, pp. 102–106.

[8] L. Lamport, Password authentication with insecure communication, CACM 24, 70–771, Nov. 1981.

[9] R, Merkle, A certified digital signature, Crypto 89, 218–238.

[10] M. Naor and M. Yung, Universal one-way hash functions and their cryptographic applications, STOC 89.

[11] NIST FIPS PUB XX, "Digital Signature Standard," National Institute of Standards and Technology, U.S. Department of Commerce, Draft, 1 Feb. 1993

[12] NIST FIPS PUB 180: Secure Hash Standard, 1993.

[13] T. Okamoto and K. Ohta, "Universal Electronic Cash," Advances in Cryptology - Proceedings of Crypto '91, 1992, pp. 324-337

[14] T.P. Pedersen, Electronic payments of small amounts, DAIMI technical report, Arhus University, Aug. 95.

[15] R. Rivest, "The MD5 Message Digest Algorithm," RFC 1321, Apr. 1992

[16] R. Rivest, A. Shamir, "PayWord and MicroMint: Two simple micropayment schemes", May 1996. (Appeared in RSA Inc. CryptoBytes).

[17] R. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, v. 21, n. 2, Feb 1978, pp. 120-126

[18] M. Sirbu and J. D. Tygar, "NetBill: An Internet Commerce System Optimized for Network Delivered Services," Compcon '95, pp. 20-25

[19] J. M. Tenenbaum, C. Medich, A. M. Schiffman, and W. T. Wong, "CommerceNet: Spontaneous Electronic Commerce on the Internet," Compcon '95, pp. 38-43