PayWord and MicroMint *(extended abstract)*

Ronald L. Rivest
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139 USA

Adi Shamir
Applied Math Department
The Weizmann Institute of Science
Rehovot 76100, Israel

Many electronic payment schemes have been proposed in recent years to address the problem of secure transactions over open networks. Micropayment schemes are a special kind of payment schemes for applications in which each payment is very small. To support micropayments, exceptional efficiency is required; otherwise, the cost of the mechanism will exceed the value of the payments.

PayWord and MicroMint are two simple micropayment schemes. In both schemes, the players are brokers, users and vendors. Brokers authorize users to make micropayments to vendors and redeem the payments collected by the vendors (See Figure 1). Broker-user and broker-vendor relationships are long-term, while user-vendor relationships are transient.

Our micropayment schemes might be considered lightweight when compared with full payment schemes in the sense that our security goal is to keep honest people honest (a similar situation exists with newspaper vending machines): exceptionally small-scale fraud and abuse cannot be totally prevented. In return, however, we try to achieve the following efficiency goals:

- Minimize the number of public-key operations by using hash operations whenever possible. (Roughly, hash functions are about 100 times faster than RSA signature verification and about 10,000 faster than RSA signature generation.)
- Minimize communication (particularly on-line communication) with brokers. Since there may be only a few nationwide brokers, it is important that their computational burden be both reasonable and "off-line."
- Make the communication per transaction efficient, especially for repeated small purchases.

...our security goal is to keep honest people honest...

PayWord is a credit-based scheme and is optimized for sequences of micropayments. It is also secure and flexible enough to support larger variable-value payments as well. MicroMint introduces a new paradigm for electronic coins and is designed to eliminate public-key operations altogether. It has lower security but higher efficiency.

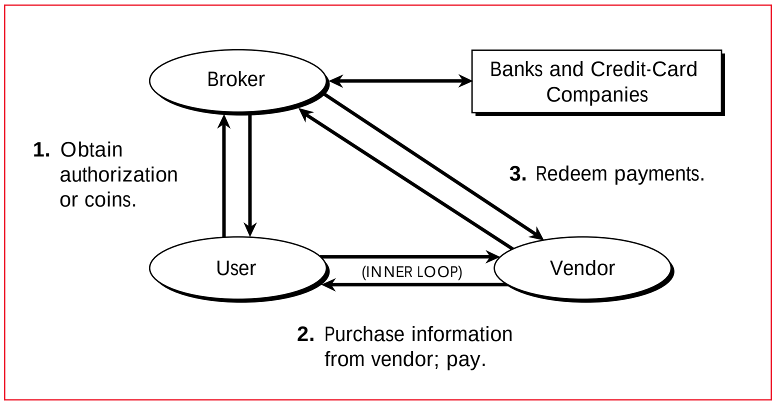
Broker
Banks and Credit-Card Companies
1. Obtain authorization or coins.
3. Redeem payments.
User
(INNER LOOP)
Vendor
2. Purchase information from vendor; pay.

Figure 1. Broker/Vendor/User Relationship

PayWord

Roughly speaking, PayWord consists of the following components:

1. A user establishes an account with a broker, who issues the user a digitally-signed certificate. This authorizes the user to make payword chains, which are chains of hash values.
2. Before contacting a vendor, the user creates a vendor-specific payword chain off-line.
3. The user authenticates the complete chain to the vendor with a single public-key signature, and then successively reveals each payword in the chain to make each micropayment.
4. The vendor redeems the paywords received from the user with the original broker.

Payword Chains

In the construction of a payword chain, we will use a one-way hash function, such as MD5 [6]. The hash function h has the property that, given a hash value y it is hard to find an input x, such that $y=h(x)$.

Ron Rivest is associate director of MIT's Laboratory for Computer Science. He can be contacted at rivest@theory.lcs.mit.edu. Adi Shamir is professor at the Applied Math Department of the Weizmann Institute of Science, Israel, and can be contacted at shamir@wisdom.weizmann.ac.il. Rivest and Shamir are co-inventor of the RSA cryptosystem.
This article is edited from the full paper [7] by Yiqun Lisa Yin. She is a research scientist at RSA Laboratories, and can be reached at yiqun@rsa.com.

A user creates a payword chain $\{w_0, w_1, w_2, \ldots, w_n\}$ by picking the final payword w_n at random and computing paywords according to $w_i = h(w_{i+1})$. The first payword w_0 is called the "root" of the chain. Given w_0 it is hard for anyone other than the user to figure out the rest of the paywords in the chain. By revealing w_0, however, the user is "committed" to the payword chain she just created.

User-Broker relationship

A user begins a relationship with a broker by requesting an account and a PayWord Certificate. The user first gives the broker over a secure authenticated channel: her credit-card number, her public key PK_U and her "delivery address" (e.g., IP-address). The broker then issues the user a digitally signed certificate which authorizes her to make payword chains until a given expiration date, and authorizes the delivery of goods only to the specified delivery address.

The user's certificate C_U thus has the following form:

$$C_U = \{\text{broker, user, user's delivery address}, PK_U, \text{expiration-date, other-info}\}_{SK_B},$$

where $\{\ \}_{SK_B}$ denotes that the contents of $\{\ \}$ are signed with the broker's private key SK_B. The PayWord certificate C_U is a statement by the broker to any vendor that authentic paywords produced by the user and used before the expiration date will be redeemed.

User-vendor relationships are transient. A user might visit a web site, purchase ten pages, and then move on elsewhere.

User-Vendor relationship

User-vendor relationships are transient. A user might visit a web site, purchase ten pages, and then move on elsewhere. When the user is about to contact a new vendor, she computes a fresh payword chain $\{w_0, w_1, w_2, \ldots, w_n\}$. Here n is chosen at the user's convenience; it could be ten or ten thousand. She then computes her commitment for that chain:

$$M = \{\text{vendor}, C_U, w_0, \text{current-date, other-info}\}_{SK_U}.$$

The commitment M, which is signed with the user's private key SK_U, authorizes the broker to pay the vendor for any of the paywords $w_1, w_2, \ldots, w_n$ redeemed on the current date. Paywords are vendor-specific and user-specific; they are of no value to another vendor. Upon receiving the commitment M, the vendor verifies the user's signature on M and the broker's signature on C_U (contained within M), and checks the expiration dates. After the commitment, payment from a user to a vendor consists of a payword and its index: (w_i, i). The payment is not signed by the user.

The user spends her paywords in order: w_1 first, then w_2, and so on. If each payword is worth one cent and each web page costs one cent, then she discloses w_i to the vendor when she orders her i-th web page from the vendor that day. This leads to the PayWord payment policy: For each commitment a vendor is paid l cents, where (w_l, l) is the corresponding payment received with the largest index. This means that the vendor needs to store only one payment from each user: the one with the highest index. The broker can confirm the value to be paid for w_l by determining how many applications of h are required to map w_l into w_0.

Vendor-Broker relationship

A vendor needn't have a prior relationship with a broker, but does need to obtain the public key of the broker in an authenticated manner, so he can authenticate certificates signed by the broker. He also needs to establish a way for the broker to pay him redeemed paywords.

At the end of each day (or other suitable period), the vendor sends the broker a redemption message giving, for each of the broker's users who have paid the vendor that day, the commitment C_U and the last payment $P = (w_l, l)$. The broker needs to first verify each commitment received by checking the user's signatures (since he can recognize his own certificates), and then verify each payment (w_l, l) by computing the hash function l times. The broker will normally honor all valid redemption requests.

Security

In a typical scenario for PayWord, each payword represents a very small amount of money, such as one cent. After the commitment, the paywords do not need to be signed by the user, since the paywords are self-authenticating using the commitment. Such a feature may allow each payword to represent a larger amount of money (e.g., software selling at $19.99). Hence, the level of security and flexibility of PayWord makes it possible to support certain "macro-payment" as well.

In PayWord, the contents of a payment does not specify what item it is payment for. A vendor may cheat a user by sending her nothing, or the wrong item, in return. The user bears the risk of losing the payment, just as if she had put a penny in the mail. Vendors who so cheat their customers will be shunned. This risk can be moved to the vendor, if he specifies payment after the document has been delivered. If the user doesn't pay, the vendor can notify the broker and/or refuse the user further service. For micropayments, users and vendors might find either approach workable.

Efficiency

PayWord is designed to minimize the on-line communication with brokers: the vendor does not need to interact with the broker when a user first contacts the vendor, nor is any interaction with the broker required as each payment is made. In PayWord, the communication per transition is also very efficient: after the commitment, each payword is only 20 to 30 bytes long. Moreover, the broker does not even receive every payword spent, but only the last payword spent by each user each day at each vendor. PayWord is thus extremely efficient when a user makes repeated requests from the same vendor, but is quite effective in any case.

PayWord's computational and storage requirements are summarized below:

- The broker needs to sign each user certificate, verify each user commitment, and perform one hash function application per payment. (All these computations are off-line.) The broker stores copies of user certificates and maintains accounts for users and vendors.

- The user needs to verify her certificates, sign each of her commitments, and perform one hash function application per payword committed to. (Only signing commitments is an on-line computation.) She needs to store her private key, her active commitments, the corresponding payword chains, and her current position in each chain.

- The vendor verifies all certificates and commitments received, and performs one hash function application per payword received or skipped over. (All these computations are on-line.) The vendor needs to store all commitments and the last payment received per commitment each day.

PayWord is thus extremely efficient when a user makes repeated requests from the same vendor, but is quite effective in any case.

MicroMint

MicroMint introduces a new paradigm for representing electronic coins by "hash function collisions." In fact, it uses hash functions only without public-key cryptography. MicroMint has the property that generating many coins is much cheaper, per coin generated, than generating few coins. A large initial investment is required to generate the first coin, but then generating additional coins can be made progressively cheaper. This is similar to the economics for a regular mint, which invests in a lot of expensive machinery to make coins economically.

In a typical setting, a broker produces coins that are valid for a given period (e.g., a month). The broker needs to invest in substantial hardware that gives him a computational advantage over would-be forgers, and run this hardware continuously for a month to compute coins valid for the next month. New coins are issued to users at the beginning of each month. Users give valid coins to vendors as payment, and vendors return coins collected to the broker at their convenience (e.g., at the end of each day).

MicroMint coins as hash function collisions

MicroMint coins are represented by hash function collisions, for some specified one-way hash function h mapping m-bit strings x to n-bit strings y. A k-way collision ($k \geq 2$) is a set of k distinct x-values ($x_1, x_2, \ldots, x_k$) that have the same hash value y.

We first give a brief review of the computation time for finding hash function collision. Assume h acts "randomly," then the only way to produce even a single k-way collision is to hash about $2^{n(k-1)/k}$ x-values and search for repeated outputs. (The case for 2-way collisions is essentially the "birthday paradox.") However, if one examines c times this many x-values, for $1 < c < 2^{n/k}$, one expects to see about c^k k-way collisions. Hence, producing collisions can be done increasingly efficiently, per coin generated, once the first collision has been passed. Note that increasing k has the dual effect of delaying the threshold at which the first collision is seen, and also accelerating the rate of collision generation once the threshold is passed.

MicroMint introduces a new paradigm for representing electronic coins by "hash function collisions."

We thus let a k-way collision ($x_1, x_2, \ldots, x_k$) represent a coin. This coin can be easily verified by checking that the x_i's are distinct and that

$$h(x_1) = h(x_2) = \ldots = h(x_k) = y$$

for some n-bit string y. Note that each coin is a bit-string whose validity can be easily checked by anyone, but which is hard to produce. This is similar to the requirements for a public-key signature, but the complexity of the signature makes it an overkill for a transaction whose value is only one cent.

Note that each coin is a bit-string whose validity can be easily checked by anyone, but which is hard to produce.

The process of minting coins

The process of computing $h(x)=y$ is analogous to tossing a ball x at random into one of 2^n bins; the bin which ball x ends up in is the one with index y. A coin is thus a set of k balls that have been tossed into the same bin. Getting k balls into the same bin requires tossing a substantial number of balls altogether, since balls can not be "aimed". To mint coins, the broker will create 2^n bins, toss approximately $k2^n$ balls, and create one coin from each bin that now contains at least k balls. In this way each ball has a chance of roughly 1/2 of being part of a coin.

A small problem in this basic picture, however, is that computation is much cheaper than storage. The number of balls that can be tossed into bins in a long computation far exceeds the number of balls that can be memorized on a reasonable number of hard disks. We thus propose to make most balls unusable as part of a coin, in a manner that depends on the hash value y. To do so, we say that a ball x is "good" if the high-order t bits of the hash value y have a value z specified by the broker. More precisely, let $n = t + u$. If the high-order t bits of y are equal to z, then y is called "good," and the low-order u bits of y determine the index of the bin in which the (good) ball x is tossed.

A proper choice of t enables us to balance the computational and storage requirements of the broker. This slows down the generation process by a factor of 2^t, without slowing down the verification process. The broker thus tosses approximately $k2^n$ balls, memorizes about $k2^u$ good balls that he tosses into 2^u bins, and generates from them about $(1/2) \infty 2^u$ valid coins.

A typical scenario

Here is a sketch of how a typical broker might choose the parameters (k, u, t, n) and make investment in hardware. We suppose that the broker wishes to have a net profit of \$1 million per month, and he charges a 10% brokerage fee to vendors for redemption. Thus, the broker needs to sell one billion coins (approximately 2^{30}) per month to collect his \$1 million fee. (This requires a customer base of 500,000 if an average user buys 2,500 coins for \$25 per month.)

The broker first chooses $k = 4$; a coin will be a good 4-way hash collision. He then chooses $u = 31$ (i.e., creates 2^{31} bins), since this will allow him to create about $(1/2) \infty 2^{31} = 2^{30}$ coins.

The broker chooses his hash function $h(x)$ as the encryption of some fixed value v with key x under DES, which can be implemented by so-called field-programmable gate array chips. Each chip costs about \$400 and computes 2^{25} values per second. Buying 2^8 of these chips costs the broker \$100,000 and allows him to compute about 2^{54} values per month. Since $k = 4$, the broker chooses $n = 52$ and $t = 21$. Storing all good pairs $(x, h(x))$ requires less than 2^{37} bytes, and costs less than \$40,000 using standard magnetic hard disk technology. Hence, the total cost for the broker's hardware is less than \$150,000, which is less than 15% of the first month's profit.

Selling coins, making payments, and redemption

At the beginning of each month, the broker either reveals a new hash function h or changes the value z for that month and sells coins to users. Such sales can be on a debit basis or a credit basis, since the broker can recognize coins when they are returned to him for redemption. In a typical purchase, a user might buy \$25.00 worth of coins (2500 coins), and charge the purchase to his credit card. The broker keeps a record of which coins each user bought. Unused coins are returned to the broker at the end of each month.

Each time a user purchases a web page, he gives the vendor a previously unspent coin $(x_1, x_2, \ldots, x_k)$. The vendor verifies that it is indeed a good k-way collision by computing k hash values. The vendor returns the coins he has collected to the broker at the end of each day. If the coin has not been previously returned, the broker pays the vendor one cent (minus any brokerage fee) for each coin. If a coin is received more than once, the broker does not pay more than once. Which vendor gets paid can be decided arbitrarily or randomly by the broker.

Security analysis

We believe that users and vendors will have little motivation to cheat in order to gain only a few cents; even if they do, the consequences are of no great concern. Our security mechanisms are thus primarily designed to discourage large-scale attacks, such as massive forgery or persistent double-spending.

Forgery: Can an adversary forge MicroMint coins? (Economically?) First, the computational difficulty of minting coins makes small-scale forgery not really a concern. (For the parameter choice given above, a forger would need to spend 80 years on a standard workstation just to generate the first coin). Second, large-scale forgers can be detected and countered. In particular, coins "expire" monthly, and the new hash function for each month is revealed only at the beginning of that month. (The broker works during May to make coins good for June; forger only learns the hash function at the beginning of June and so starts out way behind.) Additional protection against forgery may be obtained by restricting coins to satisfy "hidden predicates" which are only announced if forgery is detected by the broker. For example, legitimate coins may all satisfy condition that the low-order bit of x_i is equal to some complicated function of other bits. Forger's coins will typically not pass this additional "verification condition".

Double-spending: What if a user "double-spends" his MicroMint coins? There is no "anonymity" in MicroMint: the broker keeps track of whom each coin was sold to and notes when it is returned by vendor. Small-scale double-spending is not a concern. A user whose coins are consistently double-spent will be caught and black-listed; he will not be sold any more MicroMint coins.

Vendor fraud: What if a vendor gives copies of coins received to an accomplice? Vendors who consistently redeem coins that are also redeemed by other vendors will be black-listed and refused further redemption service by the broker. Users might cooperate with the broker to identify bad vendors by identifying where coins were first spent.

Theft of coins. If theft of coins is judged to be a problem during initial distribution to users or during redemption by vendors, it is easy to transmit coins in an encrypted form. Since user/broker and vendor/broker relationships are relatively stable, long-term encryption keys (e.g., DES keys) can be arranged.

To prevent theft of coins as they are being transferred from user to vendor, we can make coins user-specific so that they are of no value to other users. To produce such coins, we generalize the notion of a "collision" to more complicated combinatorial structures which are related to the identity of a user and can be easily checked during verification. The user-specific coins can be further modified so that they are vendor-specific as well, making a stolen coin even less desirable. Details of these extensions are in the full version of the paper [7].

Conclusions

In this article we have presented two new micropayment schemes which are exceptionally economical in terms of the number of public-key operations employed. Furthermore, both schemes are off-line from the point of view of the broker. The area of micropayments has attracted considerable attention recently, and several researchers have proposed related schemes (Millicent [4], NetBill [2], NetCard [1], Pedersen's tick payments [5], and a micropayment scheme based on iKP [3], etc). More details about our schemes and the relationships between the various proposals are described and analyzed in the full version of this extended abstract [7].

...users and vendors will have little motivation to cheat in order to gain only a few cents...

Our security mechanisms are thus primarily designed to discourage large-scale attacks, such as massive forgery or persistent double-spending.

References

[1] R. Anderson, H. Manifavas, and C. Sutherland. NetCard – A practical electronic cash systems. 1996. Available from author: Ross.Anderson@cl.cam.ac.uk.

[2] B. Cox, J.D. Tygar, and M. Sirbu. NetBill security and transaction protocol. http://www.ini.cmu.edu/netbill/home.html.

[3] R. Hauser, M. Steiner, and M. Waidner. Micro-payments based on iKP. January 1996. http://www.zurich.ibm.com:80/Technology/Security/extern/ecommerce/iKP.html.

[4] M. Manasse. The Millicent protocols for electronic commerce. 1995. http://www.research.digital.com/SRC/millicent.

[5] T. Pedersen. Electronic payments of small amounts. Technical Report DAIMI PB-495, Aarhus University, Computer Science Department, August 1995.

[6] R. Rivest. RFC 1321: The MD5 Message Digest Algorithm. RSA Data Security, Inc., April 1992.

[7] R. Rivest and A. Shamir. PayWord and MicroMint– Two simple micropayment schemes. April 1996. http://theory.lcs.mit.edu/~rivest.