Information Systems Frontiers 7:4/5, 371–389, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

e-coupons*: An Efficient, Secure and Delegable Micro-Payment System*[*]

Vishwas Patil and R.K. Shyamasundar
School of Technology and Computer Science, Tata Institute of Fundamental Research, Colaba,
Mumbai - 400005, India
E-mail: vtp@tifr.res.in
E-mail: shyam@acm.org

Abstract. *In this paper, we propose a new efficient and secure micro-payment scheme, named* e-coupons*, which can provide the users the facility of delegating their spending capability to other users or their own devices like Laptop, PDA, Mobile Phone, and such service access points. The scheme has the promise of becoming an enabler for various Internet-based services involving unit-wise payment. It gives flexibility to the users to manage their spending capability across various access points for a particular service without obtaining an authorization for each and every access point from a facilitating bank. This flexibility which is not present in the existing micro-payment schemes is essential for accessing ubiquitous e-services and other Internet-based applications. The facility of delegation introduces a slight overhead in respect of the proof or verification of the delegated authorization and security provided to the payments. The payoff from the facility of delegation takes away the burden of the overhead. The paper discusses the design of the protocol and provides a basic analysis of the performance of the system.*

e-coupons *is based on PayWord, a single-seed one-way hash chain for unit-wise payment, TESLA for payment security and SPKI/SDSI as underlying PKI framework for its unique delegation feature. The results obtained from the implementation of* e-coupons *are quite acceptable and show near real-time response. Our scheme uses multi-seed one-way hash chains for unit-wise payment. Furthermore, it allows an ordered transfer of the portions of payment chains to others. Because of this user's spending capability can be used from different service access points to access the subscribed service, concurrently.*

Key Words. *micro-payment, security, delegation, one-way hash function, PayWord, TESLA, SPKI/SDSI*

1. Introduction

E-Commerce covers a broad spectrum of transactions varying from macro-transactions to micro-transactions. In macro-transactions, while the value of each transaction is very high, the challenge lies in providing a higher grade of authentication, payment security, and non-repudiation of transactions. In case of micro-transactions, while the need is to cater to a large volume of transactions of low intrinsic financial value, the challenge is to keep the cost of each transaction to a minimum on an average.

Micro-transactions include Internet-enabled services like streaming multi-media, accessing computational power from grids, loadable softwares, software plug-ins/APIs, VoIP calls, e-Library, news, and various such non-tangible goods which can be delivered through Internet (of them, news is free, for example). Subscribers access such services through different service access points and would not always like to reveal the set of their access points. The service providers have not succeeded in charging their services by following the available means. Hence, they provide the services to the users free of cost or employ mechanisms other than a micro-payment system. The service providers recover the cost from the advertisers or bulk subscriptions using authentication based on host IP addresses and/or browser cookies etc. A direct micro-payment mechanism would be of great complement and facilitate small vendors. Further, it would provide incentives for sporadic users who do not want a full-time subscription to some paid service. Unlike macro-payments, the monetary value of every micro-payment is extremely low and the risk involved is acceptable. While the macro-payments emphasizes on security, non-repudiation and

Present address: Vishwas Patil, Dipartimento di Informatica, Università degli Studi di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy. E-mail: patil@di.uniroma1.it
*This work is supported by grants from Ministry of Information and Technology, Government of India.

atomicity of the transaction, micro-payment systems aim at efficient, low-cost, secure setup. The users are ready to accept micro-payment systems with reasonable risk factors associated with it. In this paper, we are concerned with the design and development of a micro-transaction system that charges the user directly complying with requirements such as security, low-cost per transaction, and delegation facility.

There are several e-payment schemes proposed in the literature: PayWord and MicroMint, (Rivest and Shamir, 1996); MilliCent, (Glassman et al., 1995); MiniPay, (Herzberg and Yochai, 1997); NetBill, (The NetBill Electronic Commerce Project, 1995); NetCard, (Anderson et al., 1996); NetCash, (Medvinsky and Neuman, 1993); Agora, (Gabber and Silberschatz, 1996); MPTP, (Hallam-Baker, 1995); *i*KP, (Bellare et al., 2000) based micro-payment, etc. These schemes can be broadly differentiated into *on-line* and *off-line* categories based on the type of payment validation used. In *off-line* methods, risk naturally arises as immediate validation is not performed. NetBill, NetCash, MiniPay, MilliCent use *on-line* or *semi-online* type of payment validation, which is costly in general. NetBill is designed for buying information goods via Internet with emphasis on security and atomicity of transactions. A central trusted server is involved in every transaction. Micro-economy and scalability are questionable because of the extensive network traffic required during the transaction and the interaction with the central NetBill server. NetCash is another *on-line* scheme that offers a framework for a secure and partially anonymous real time digital payment system. The basic NetCash structure consists of independent, distributed currency servers providing a link between anonymous electronic currency and non anonymous services. Currency server provides customer services, like double spending detection, coin exchange to allow untraceability, purchases of coins with cheques and redemption of coins for cheques. MiniPay features a low cost, negligible delay, natural user interface, scalable design, support for multiple currencies, and high security—including non-repudiation, overspending prevention, and protection against denial of service. MiniPay architecture involves four to six parties in its setup. It uses public-keys to authenticate parties and it is based on peer to peer relationships, where public-keys are exchanged and authenticated using existing relationships between the peers. MilliCent is a proprietary voucher based digital micro-commerce system. The system uses merchant specific vouchers, called scrip, a form of token that is only valid with a particular merchant for a limited period of time. MilliCent transactions are not anonymous and mostly *off-line*. PayWord is an *off-line*, extremely efficient, credit-based micro-payment scheme. It is a tripartite scheme involving a bank, the vendors and users. The bank gives credit facility to the users and assures the vendors for redemption of payments made by the registered users. The other micro-payment schemes; micro-iKP, NetCard, MPTP are largely based on PayWord proposal. We have excluded from our discussion the other micro-payment schemes (like Mondex, CAFE) that rely on special hardware like smartcard.

The above micro-payment schemes have been designed with the intention to make secure and/or efficient payments for non-tangible goods on *pay-per-view/pay-per-click/pay-as-you-go* basis. The schemes either rely more heavily on asymmetric key applications or they are more burdensome for the bank in terms of minting coins and *on-line* verification. Furthermore, most of these proposals are not scalable due to their centralized design. The present day applications demand much more than what these schemes provide. Nowadays, the users employ software robots to make purchases on their behalf. The users expect their subscribed services to be accessible from different access points (plausibly simultaneously). We present two typical micro-payment scenarios that cannot be satisfactorily handled by the existing schemes discussed above.

> *Scenario 1* A consortium of academic institutions has subscribed itself to various electronic publication services. The consortium management is willing to lure other non-member institutions for these subscriptions on an ad-hoc or permanent basis. For this purpose, it is necessary for the consortium administrator to have features like partial delegation of authority to other institution or individual while maintaining the efficiency and security of the system.
>
> *Scenario 2* A user has a multi-threaded application and the threads make use of different external APIs/plug-ins (Application Program Interfaces), based on the subscriptions of the user. A monolithic payment instrument will hinder the execution of threads in parallel.

One of the principal reason is that the existing micro-payment schemes do not allow a user to delegate his authority totally or in part to third parties. There are many such scenarios where the growth of e-commerce has stagnated due to unavailability of an efficient and secure micro-payment scheme that provides delegation facility to users over their spending capability. In this paper, we shall address the design and implementation of a micro-payment system satisfying these requirements: security, low-cost per transaction, efficiency, and a provision to delegate the spending capability. Our design uses features from PayWord (Rivest and Shamir, 1996), TESLA (Perrig et al., 2002; Perrig et al., 2001), and SPKI/SDSI (Clarke et al., 2001). In other words, we have extended the PayWord framework (Rivest and Shamir, 1996) to handle delegation of users' spending capability through SPKI/SDSI and handling security through TESLA.

Our scheme is a credit-based and *off-line* scheme. Also, the coins/paywords (payment's primitive unit) are vendor-specific and not user-specific, unlike PayWord. It is necessary to keep the coins only vendor-specific and not user-specific as coins are going to change hands. One needs to provide security to coins since there is a threat that they can be snatched in transit and submitted in real-time to the vendor. We employ the modified TESLA protocol for this purpose. It not only provides an efficient method of source authentication but also provides economical security to the coins and thwarts the *man-in-middle* and *denial-of-service* attacks. We make use of SPKI/SDSI (Clarke et al., 2001; Ellison, 2002) framework as a Public-Key Infrastructure satisfying the requirements (especially delegation of authorization) of the parties involved in our setup. Our implementation is efficient, secure and capable of addressing the exemplary micro-payment scenarios described above with ease of manage-ability and maintenance.

Rest of the paper is organized as follows: In the next section, we provide an overview of protocols in isolation i.e., PayWord, TESLA, and SPKI/SDSI followed by our proposal in Sections 3 and 4. Section 7 gives a detailed analysis of our protocol in terms of security aspects, risk factors and performance. The paper concludes with a discussion in Section 8.

2. Overview

In this section, we provide a brief overview of the schemes such as PayWord, TESLA and SPKI/SDSI on which our scheme is based.

2.1. PayWord

It is a credit-based, *off-line* micro-payment scheme, that uses chains of *paywords* (one-way hash values representing primitive monetary units). The thrust of the scheme lies in minimizing the number of public-key operations required per payment and thereby achieving exceptional efficiency (Rivest and Shamir, 1996). It is a tripartite mechanism involving a user U who makes the payment, a vendor V who receives the payment and a broker B (a financial intermediary) who keeps accounts for the parties concerned. Broker is a trusted party and gives credit facility to the users for transacting with the vendors. After reaching a formal credit agreement between B and U, B promises V to redeem the paywords spent by U at regular intervals of time.

Before making any payments to the vendor, the user generates a payword chain which is user-specific and vendor-specific. The user generates the payword chain in reverse order by picking the last payword w_n at random, and then subsequently computing each payword $w_i = h(w_{i+1})$ for $i = n - 1, n - 2, \ldots, 0$, where h is a strong hash function and w_0 is called the *root* (*commitment*) of that payword chain. The user has to register such payword chains with the vendor before using the chains as a payment instrument. The user submits the payword chain's *commitment* value (w_0) along with the authorization (PayWord certificate) that empowered the user to generate such a payment instrument.

On successful verification on vendor's side, the user can use the registered payword chain for unit-wise buying activity. While making the unit-wise payments using the generated paywords, the i-th payment (for $i = 1, 2, \ldots, n$) from U to V consists of the pair (w_i, i) which V can verify using w_{i-1} with the help of the one-way hash function, h. Each such payment requires no computations by the user, and only a single hash operation by the vendor for verification. The vendor can verify the payment by computing the hash of the present payword and checking that it is equal to the prior payword respective the root in the commitment for the first payword tendered.

For redemption of the accumulated paywords, at regular intervals the vendor interacts with the facilitating bank and reports the last (highest-indexed) payment (w_l, l) received from each registered user after last such reporting, together with each corresponding *commitment*. On verification, the bank charges user's account l units of currency and deposits it to vendor's account. Note that it is therefore unnecessary for the bank to maintain large online databases.

Relationship between Bank, User and Vendor: Let the public-keys of bank B, user U, and vendor V be denoted by K_B, K_U, K_V and their private-keys be denoted by K_B^{-1}, K_U^{-1}, K_V^{-1} respectively. The interaction between the three parties is described below:

B $\leftrightarrow$ U: User U approaches B with its delivery-address details (A_U) and some additional information (I_U) for obtaining the PayWord certificate $C_U = \{B, U, A_U, K_U, E, I_U\}_{K_B^{-1}}$ where E is the certificate expiry date i.e., the date up to which the subscribed service can be availed.

U $\leftrightarrow$ V: U computes a payword chain $w_1, \ldots, w_n$ with root w_0 and then it generates a *commitment* for the payword chain: $M = \{V, C_U, w_0, D, I_M\}_{K_U^{-1}}$ where D is the current date and I_M is some additional desired information. A payment $P = (w_i, i)$ from U to V consists of a payword and its index i.

V $\leftrightarrow$ B: At regular time intervals, vendor V redeems the accumulated paywords with bank B. In each such redemption request, V produces every subscriber's payword chain *commitment* with the respective C_U received from subscriber U (if it has not already done so in previous redemption requests), and the last payment $P = (w_l, l)$ received from each user. On verification of the received signed commitments, B does the accounting work i.e., it deducts l units from U's account and credits it to V's account. This payment settlement takes place outside the PayWord system.

PayWord is optimized for sequences of micro-payments, but is secure and flexible enough to support larger variable-value payments as well, depending upon how much risk the bank and vendor are willing to take. The scheme has user-specific, vendor-specific payword chains and hence, an adversary has no interest in either stealing it while in transit or to double-spend. As a consequence, PayWord cannot provide anonymity to the transactions. When the user requires multiple payword chains for its own use e.g., for accessing subscribed services via multiple devices (PC, Laptop, PDA, Mobile Phone, etc.,) it has to request and register separately for each device, which makes the system inefficient since it increases the costly initial interactions with the bank.

2.2. TESLA

TESLA (Timed Efficient Stream Loss-tolerant Authentication) is an efficient source authentication protocol with low communication and computational overhead (Perrig et al., 2002, 2001). It uses pure symmetric cryptographic functions (MAC–Message Authentication Code (Schneier, 1996) functions) and achieves asymmetric properties through loosely synchronized clocks and delayed key disclosure. It uses the time difference between the sender and receiver for achieving asymmetry.

The TESLA protocol is briefed in the following: The sender of the message attaches the MAC over each outgoing packet calculated using a key k which is known only to the sender. The receiver goes on buffering such packets and authenticates them as soon as the sender discloses k in its subsequent transmissions. At regular intervals, the sender changes key k used for MAC computation. These values of k are derived from a one-way collision resistant hash function in such a way that the subsequent values can be authenticated in reverse order. Due to such use of one way hash chain values for computing MAC over outgoing packets, the receiver can thwart the *denial-of-service* and *replay attacks* by simply looking at the packet time-stamp, the key disclosed by the sender at that time, and can ignore dubious packets (Perrig et al., 2002). The original protocol is briefly described below.

Before starting the actual transmission, the receiver and sender loosely synchronize their time. During this process, receiver is interested in calculating the *maximum time synchronization error* Δ. The receiver records its local time t_R and sends a *Nonce* as a time synchronization request to the sender. The sender responds with a digitally signed message $\{t_S, Nonce\}_{K_S^{-1}}$, where t_S and K_S^{-1} are sender's local time and private-key respectively. On successful verification of the $Nonce$ returned by the sender, the receiver computes the upper bound on the sender's current time as $t_s \leq t_r - t_R + t_S$, where t_r is receiver's current time. After this process, the actual time synchronization error δ, that is the difference between the sender and the receiver's time, is computed.

Now the sender splits up the time into intervals of uniform duration and assigns the values of a *one-way* hash chain [cf. Appendix A] sequentially to each time interval to generate MACs over packet data during the respective time intervals. Sender defines a disclosure time d for *one-way* chain values and conveys it to the receiver. On receiving the packets appended with MACs computed over it by the sender, the receiver performs the following: Since the schedule for disclosing the keys are known and the clocks are loosely synchronized, the receiver can check that the key used to compute the MAC is still secret by determining that the sender could not have yet reached the time interval for disclosing it. If the MAC key is still secret, then the receiver buffers the packet. The sender sends the most recent *one-way* chain value that it can disclose with each packet; the receiver checks that the disclosed key is correct by virtue of the property of *one-way* chains, and then checks the correctness of the MAC of buffered packets that were sent in the time interval of the disclosed key. The receiver accepts the packet only if the MAC sent by the sender matches with its locally computed value.

TESLA has low computation overhead for the generation and verification of authentication information, and has low communication overhead. Limited buffering is required for the sender and the receiver, hence timely authentication for each individual packet. It uses delayed disclosure of encryption key and achieves the property of data confidentiality and authentication efficiently, which is generally provided by asymmetric cryptographic methods. TESLA cannot provide non-repudiation, an important requirement for financial transactions. Security of the TESLA also relies on the fact that earlier keys become redundant after a period of time.

2.3. SPKI/SDSI

SPKI and SDSI were two separate efforts initiated to overcome the complexity, privacy and trust related issues faced by the traditional highly centralized PKIs (Ellison, 2002). Later these schemes were merged and called SPKI or SPKI/SDSI. It uses *s-expressions* to represent the data structures, which provides the user much needed transparency and avoids ASN.1 (Abstract Syntax Notation One (ASN.1,)) encoding. Unlike the global naming scheme employed in hierarchical PKIs, it uses local name spaces associated with each public-key. So every principal can issue/define the key bindings locally. Principals can create new definitions binding other principal's keys or names based on the trust he is willing to put, like PGP's *web-of-trust* (PGP,). This scheme follows the bottom-up approach, unlike X.509's top-down approach (Ford and Baum, 2002), and has provisions to accommodate global root Certification Authorities (CAs). Also, the separation of authorization from naming prevents unnecessary revelation of user's authorizations which are not required while executing a particular authority. This is not possible when certificates play naming and multiple authorization bindings together. Furthermore, the threshold certificates and group certificates allow a security administrator to write the access control policies in a manageable way [cf. Appendix A]. A brief functional outline of the usage is illustrated with the following scenario:

Let principal K serve as a resource provider denoted by RESOURCE *and specify the ACL for its access. K authorizes principals K_1, K_2, K_3 to act as retailers for its service. A principal K_s subscribes for the* RESOURCE *service via one of the retailers. Let us see how the subscriber K_s comes up with an authorization proof to access* RESOURCE*, and how the resource owner K makes use of group certificates and extended names to efficiently specify and manage the access to the resource.*

Design for such a policy is given in the following using SPKI notations.

- $K\ \ my\_retailers \longrightarrow \{K_1, K_2, K_3\}$ is a "$my\_retailers$" group defined by principal K and it can enforce some common policy on all the three subject principals by just narrating the policy over the name definition my_retailers.
- $K\ my\_customers \longrightarrow \{K_A, K_B, K_3\ customers\}$ is another local group definition by principal K, where it has included another group i.e., K_3's $customers$ apart from K_A and K_B, where $K_3\ customers \longrightarrow \{K_p, K_q, K_r, K_s\}$.
- Principal K consolidates its groups and creates a new definition, $K\ \ my\_groups \longrightarrow \{K\ \ my\_retailers, K my\_customers\}$ and empowers its members to access the RESOURCE by making an authorization definition, K RESOURCE$\longrightarrow K\ my_groups\ \square$, where the $\square$ (*live* delegation flag) indicates further delegation of authority is allowed to the subject principals.
  The sequence of messages, between RESOURCE controller K and requester K_s is given below:
- K_s sends an access request for RESOURCE. Controller K demands K_s to satisfy the access control policy enforced by rule K RESOURCE $\longrightarrow K\ my\_groups\ \square$.
- K_s requests for the definition of K's my_groups.
- K provides the definitions of its groups my_groups, $my\_retailers$, and $my\_customers$.
  In K's $my\_customers$ definition, K_s finds the missing authorization link.
- Proof of K_s's certificate chain discovery is:
  K RESOURCE $\longrightarrow K\ my\_groups\ \square$
  K RESOURCE $\longrightarrow K\ my\_customers\ \square$; since
  $\underline{K\ my\_groups} \longrightarrow$
  $\{K\ my\_retailers, \underline{K\ my\_customers}\}K$ RESOURCE$\longrightarrow$
  $K_3\ customers\ \square$; since

$\underline{K\ my\_customers} \longrightarrow \{K_A, K_B, \underline{K_3\ customers}\}$
and
$\underline{K_3\ customers} \longrightarrow \{K_p, K_q, K_r, \underline{K_s}\}$
$\therefore\ K_{\text{RESOURCE}} \longrightarrow K_s\ \square$

In this manner, K_s proves its access credentials over RESOURCE and is capable of delegating the authority further. But a ■ (*dead* delegation flag) in the access control definition of RESOURCE will restrict K_s from further delegation.

Such a distributed security infrastructure facilitates in designing and efficiently managing complex security models. Its ability to allow users to locally define their own name and authorization binding helps in achieving natural trust models, which are not rigidly dependent on global root CAs. The existing micro-payment schemes seem to imply reliance on a centralized certification authority infrastructure, which is facing scalability problems and has hierarchical trust relationships.

3. e-coupons: Basic Scheme

In this paper, our primary concern is the design and implementation of a micro-payment system with the following features:

1. The system should at least be on par with the PayWord system, in terms of efficiency and security,
2. It should allow users to delegate their spending capability.

The heart of our construction is a vendor-specific PayWord protocol enabled with TESLA assisted source authentication and confidentiality mechanism for the paywords (coins). Though our implementation is similar to PayWord in spirit, instead of generating a single payword chain and spending it over the time period, user generates multiple payword chains and use a statistical management approach (which varies from user to user based on their spending patterns) for spending it over non-conflicting time intervals. The SPKI/SDSI framework not only provides properties such as *non-repudiation*, but also provides the important feature of delegation. Our protocol is an *off-line* protocol, which is a very important feature of any micro-payment scheme.

In the following, we shall describe the basic protocol without the feature of delegation for the sake of clarity. The protocol's delegation feature is separately explained in Section 4. The transactions of the basic protocol among the three parties is described in Fig. 1, and the details of each transaction step are described below:

ⓐ ***Request for PayWord certificate.*** Using standard payment protocol the user establishes a session with the bank and chooses an appropriate mode for payment. Credit card information CC_U, user's public-key K_U and other supporting information I_U is sent to the bank using a standard payment protocol. The details of the user needed by the bank for the transaction and the mechanism employed are not of relevance here.

ⓑ ***Issuance of PayWord certificate.*** Based on user's credit worthiness, bank denies or issues a PayWord certificate C_U to the user U. This enables the user to generate the payword chains locally, for which the bank's guarantee of redemption exists for the paywords spent by the user with a vendor. The signed reply contains the relevant trust building information for the vendor, i.e., PayWord issuing bank B, user's name, public-key, certificate expiry time E and other information I_U (payword chain limit).
Upon receiving the PayWord certificate, user mints its' paywords by choosing a random number r and applying a standard, collision resistant, cryptographically secure *one-way* hash function h, such as SHA-1 (NIST, 1995), over it successively over a range specified by the bank in the PayWord certificate. i.e., $h(r) = w_n, h(w_n) = w_{n-1}$ for $n + 1$ times and $h(w_1) = w_0$; w_0 is called the *root* or *commitment* of the payword chain, which itself is not used as a payword (cf. Fig. 2).

ⓒ ***Registration with vendor.*** This is a one-time process in which user sends a digitally signed message containing vendor's name V, its PayWord credentials C_U, payword chain $root/commitment\ w_0$, and optional information I (what length of substring from w is used as payword encryption key e.g. 56-bit or 64-bit). Vendor verifies user's signature and authenticity of the PayWord certificate C_U enclosed in M. Vendor also checks for the presence of the $commitment$ value provided by the user in its registration entries to thwart the double-spending effort.

ⓓ ***Time synchronization request.*** Before giving the user a *go-ahead* signal, the vendor time synchronizes itself with the user so that it can weed out the fake packets from its buffer and authenticate the source of remaining packets. This is the first step in initializing TESLA. The vendor records its local time

U	$\longrightarrow$	$\langle CC, K_U, I_U \rangle_{SSL}$	$\longrightarrow$	B	…ⓐ Macro-Payment for PayWord Certificate
U	$\longleftarrow$	$C_U = \{B, U, K_U, E, I_U\}_{K_B^{-1}}$	$\longleftarrow$	B	…ⓑ User receives the PayWord Certificate
U	$\longrightarrow$	$M = \{V, C_U, w_0, I\}_{K_U^{-1}}$	$\longrightarrow$	V	…ⓒ Registration of PayWord chain with Vendor
U	$\longleftarrow$	$\langle Nonce \rangle_{K_V^{-1}}$	$\longleftarrow$	V	…ⓓ Time Synchronization Request
U	$\longrightarrow$	$\langle t_U, Nonce, T_{int}, d \rangle_{K_U^{-1}}$	$\longrightarrow$	V	…ⓔ Response and Agreement
U	$\longrightarrow$	$\{\langle w_i \rangle_{w_i}, MAC(w_i), w_{i-d}\}$	$\longrightarrow$	V	…ⓕ Micro-Payments
U	$\longrightarrow$	$\{\langle w_{i+1} \rangle_{w_{i+1}}, MAC(w_{i+1}), w_{i+1-d}\}$	$\longrightarrow$	V	…ⓖ ↓
U	$\longrightarrow$	$\{\langle w_{i+2} \rangle_{w_{i+2}}, MAC(w_{i+2}), w_{i+2-d}\}$	$\longrightarrow$	V	…ⓗ ↓

Fig. 1. e-coupons *Protocol stages.*

$\longmapsto$ *generation sequence* $\longmapsto$ $\longmapsto$

$w_n, w_{n-1}, w_{n-2}, \ldots \ldots w_{200}, w_{199}, w_{198} \ldots w_2, w_1, w_0$

$\longleftarrow$ $\longleftarrow$ *spending sequence* $\longleftarrow$

Fig. 2. *PayWord chain generation.*

t_R and sends a $Nonce$ encrypted with the private-key as a secure time synchronization request.

ⓔ ***Reply.*** In response to the time synchronization request from the vendor, the user sends an encrypted message consisting of its local time t_U, $Nonce$, the time interval T_{int} for which one key will be used for encryption, and integer d that signifies time intervals for which the key will remain secret. After successfully verifying the value of $Nonce$ returned by U, vendor computes the *maximum time synchronization error* Δ as explained in Section 2.2.

ⓕ,ⓖ,ⓗ ***Secure Payment.*** Now, the vendor is ready to accept the payword. Instead of sending the paywords in plain format, user encrypts each payword with the payword itself as an encryption key i.e., $\langle w_i \rangle_{w_i}$. The encrypted payword is sent with its MAC, computed using the payword as the key, and the most recent key which the user can reveal. For the first d messages the user does not have to reveal any key. From $(d+1)^{st}$ message, the user will start disclosing appropriate keys. Therefore, the vendor buffers the last d messages and authenticates the first payword as soon as the key is disclosed by the user in $(d+1)^{st}$ message. So, every payword verification is delayed by d intervals, which is generally a small integer value. By employing the TESLA mechanism, vendor authenticates the packet and for verification of the payword, it applies the hash function h over the payword and checks it against the last payword submitted by the user. Thus, the authentication and verification of payword goes hand-in-hand with a small delay d.

4. e-coupons*: Scheme with Delegation*

Delegation is an important feature which is missing in existing micro-payment schemes since users have different devices to access a subscription service. It is obviously not advisable to register all possible devices with the vendor and have separate prior agreements with bank. Our aim should be to minimize the costly computations particularly for hand held devices with limited resources. Our scheme allows a user to register from her PC and delegate the spending capability to her own devices or even to other users. We achieve delegation of spending capability through multi-seed payword chains using the SPKI/SDSI authorization certificates without burdening further with the PKI operations.

In the following, we shall highlight the way delegation is integrated in *e-coupons*. For this, let us assume the following authorization certificate.

$$K_U\ Chain1_{[201-402]} \longrightarrow K_{agent3} \blacksquare$$

Through this authorization certificate, the principal K_U

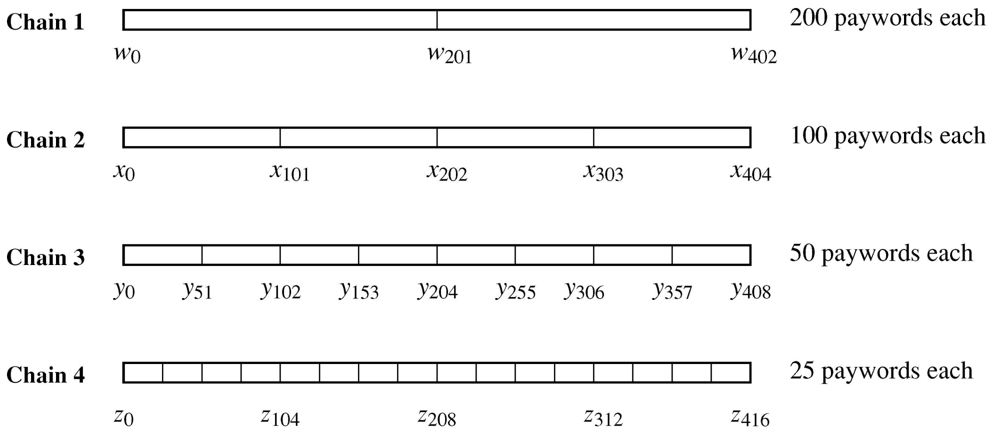

Fig. 3. *Multi-seed payword chains generation.*

delegates its authority over a portion of *Chain1* to K_{agent3}. The *dead* delegation flag in the certificate implies that K_{agent3} can exercise the authorization but cannot further delegate it to others.

So, in our *e-coupons* system, a user willing to make use of delegation facility starts requesting the bank for a PayWord certificate and also notifies the bank about its multi-seed payword generation requirements so that the bank anticipates more than one *registration* by the user with a vendor using multiple values from a payword chain as *commitments*. The bank issues the PayWord certificate to the user and the user starts generating the payword chains. A sample schematic presentation of the multi-seed payword chains is shown in Fig. 3, and some portions of these multi-seed payword chains are delegated to three different *agents* as follows:

$$\text{User} \longrightarrow \{w_0, K_{agent1}, E, I_U, \langle w_l \rangle_{K_{agent1}}\} \longrightarrow \text{agent1}$$
$$\text{User} \longrightarrow \{x_0, K_{agent2}, E, I_U, \langle x_{100} \rangle_{K_{agent2}}\} \longrightarrow \text{agent2}$$
$$\text{User} \longrightarrow \{w_{201}, K_{agent3}, E, I_U, \langle w_{402} \rangle_{K_{agent3}}\} \longrightarrow \text{agent3}$$

User authorizes its software agents to spend on its behalf by issuing an authorization certificate consisting of the following information: *commitment* for the agent, spending limit, expiry date, agent's public-key and other application specific data.

These agents/sub-users do not have to randomly choose a number and compute their own chain, but the start and end values of the chain will be provided by U. The *root* value is enclosed in the certificate and the upper ceiling value (w_l) is sent to the *agent1* in encrypted format, i.e., $\langle w_l \rangle_{K_{agent1}}$. *agent1* starts applying the hash function h over w_l for l times and reaches the *root* value defined by U. As soon as the User delegates such an authority over the part of payword chain, it locks that chain from further access until the portion of the chain is fully spent or expired. So, the next payword request from another sub-user will be served from a different un-locked payword chain. Thus, the User's ability to delegate partial authority over payword chain is restricted by the number of un-locked payword chains with the User. The agents/sub-users having authority to spend paywords from different payword chains, can transact concurrently. If the User has a priori knowledge about the pattern of requests coming from the sub-users, it can intelligently partition the payword chains of well-calculated length.

The registration step ⓒ will be a little different, that is

$$\text{agent3} \longrightarrow \{V, X, w_{201}, I_{agent3}\}_{K^{-1}_{agent3}} \longrightarrow \text{Vendor}$$

where X is *agent3*'s proof of authorization in SPKI/SDSI, and w_{201} is the *root* value of the payword chain which *agent3* is going to spend. Vendor verifies signatures over the registration message and checks authenticity of the proof presented by the requester and proceeds further.

At periodic intervals, vendor submits all the signed commitments to the bank with corresponding highest spent payword value and its index. Bank verifies this data provided by the vendor *off-line* and accordingly credits money to vendor's account from user's account.

Before considering the analysis of our micropayment scheme *e-coupons*, we shall summarize the obligations of the actors in the protocol, namely the Bank, the Vendor, and the User.

Bank is the trusted party in this setup. It acts as a facilitator for Vendors and Users. Banks enter into legal

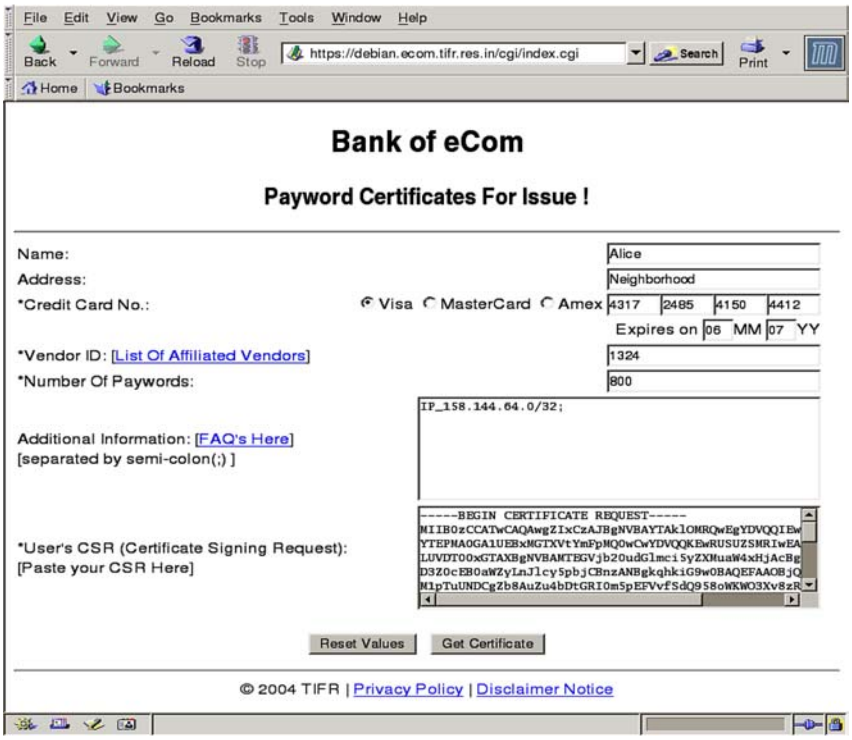

Fig. 4. *Bank's interface to issue PayWord certificates to users.*

agreement with these two other entities independently, under which they agree for honest behavior. Bank guarantees the participating Vendors for redemption of the paywords spent by the registered Users. Vendors are bound to deliver the goods on receiving the agreed payment amount. Users are required to spend their payword chains in the reverse order of its creation and they are responsible for managing their own multi-seed payword chains, and risk to lose the paywords by not following the order in which they have to be spent.

5. e-coupons*: Implementation*

In this section, we shall provide the first hand implementation perspective of our system followed by few snapshots explaining system's interface for its users. The development of *e-coupons* is done on three GNU/Linux-based Intel-Pentium machines holding the Bank, Vendor, User programs each and a laptop, as an additional service access point for the user, to demonstrate user's delegation of paywords from her PC to laptop.

The software development of *e-coupons* is an integration of three main modules; namely PayWord, modified TESLA, and SPKI/SDSI. The implementation of these modules is done using C, Perl-CGI, and other scripting languages, mySQL as database for Bank and Vendor, Apache as a web server for Bank, and OpenSSL libraries for cryptographic primitives. We have described the setup of individual entities of *e-coupons* in the following:

- **Bank:** It is central to the system. This trusted entity vouches for the users by issuing certificates. A SPKI/SDSI module runs on the Bank machine acting as CA. Bank's interface for users is a wrapper built around Matt Fredette's (MIT, Cambridge) `sdsi2sh` tool. This interface is hosted by SSL-enabled Apache server that accepts input from users and issues PayWord certificate by accepting payments by credit-cards. The input collected from user is directed to `sdsi2sh` shell by Perl-CGI scripts. The s-expression based certificate is presented to user, while a copy is stored in Bank's database. Fig. 4, gives a glimpse of User's interaction with Bank. Bank's server listens on port 2344 for redemption requests from Vendors.

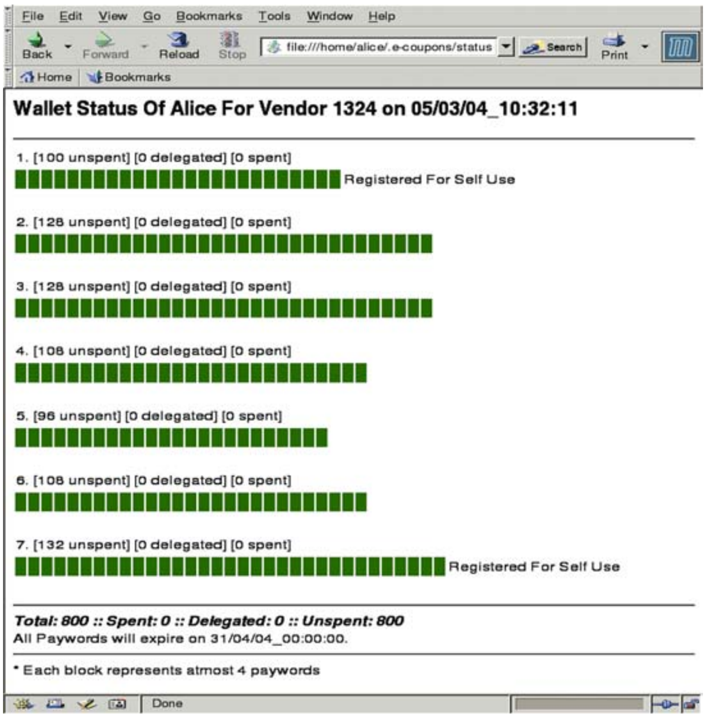

Fig. 5. *User's wallet interface: initial status at time* t_0.

This program accepts data (user commitments and/or payword indices) from Vendors and stores the commitments to respective Vendor's database table and redeems the Vendor's account according to the payword index after validating its value. These computations are done with less priority depending upon the load on server.

- **User:** After making macro-payment for a particular service of a Vendor, user receives the authority, via PayWord certificate, to mint its own paywords for payment purpose. This certificate is stored into User's `$HOME/.e-coupons/certificates/` directory with name `$VENDORID.crt`, by default. Users are provided with an interactive program that helps them to generate payword chains of different lengths based on their requirements. The program checks for absence of a lock file `$VENDORID.lock` in `$HOME/.e-coupons/status/used-certs/` directory before generating the payword chains for a particular Vendor. Upon successful run it generates a lock and stores the generated chains in separate files under `$HOME/.e-coupons/chains/$VENDORID/` directory. Taking into consideration the space required for storing paywords into a file and the time required for accessing each payword from the file versus generating the payword value afresh having readily available previous value in CPU cache, our program just stores the intermediate values and corresponding index into the payword file apart from the chain's boundary values. Having generated the paywords (Fig. 5, shows User's initial wallet status), the User can either register the chains for self use or delegate the chains, in totality or partially, to others. For using a chain for self use, the User signs the chain's root value as its commitment and registers it with the vendor after initializing the TESLA server on port 2345 for handling time-synchronization requests arising from Vendor side. User is also equipped with the `sdsi2sh` shell for delegating its spending capability. Fig. 6, shows the usage of utility provided to users to delegate the spending capability. A full-fledged interface is provided to users of the system to manage the wallet. Figs. 7, 8, 9, shows User's wallet status at different time intervals highlighting the paywords in distinct color codes indicating "unspent", "delegated" and "spent" paywords.
- **Vendor:** In our implementation, we have integrated a streaming music server, as Vendor's service, that listens on port 6666 for User requests. Vendors

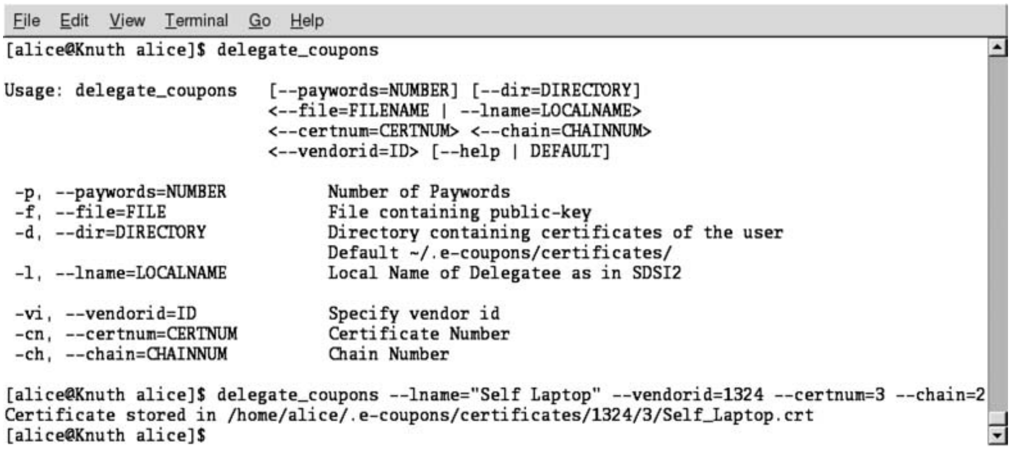

Fig. 6. *User delegating its spending capability.*

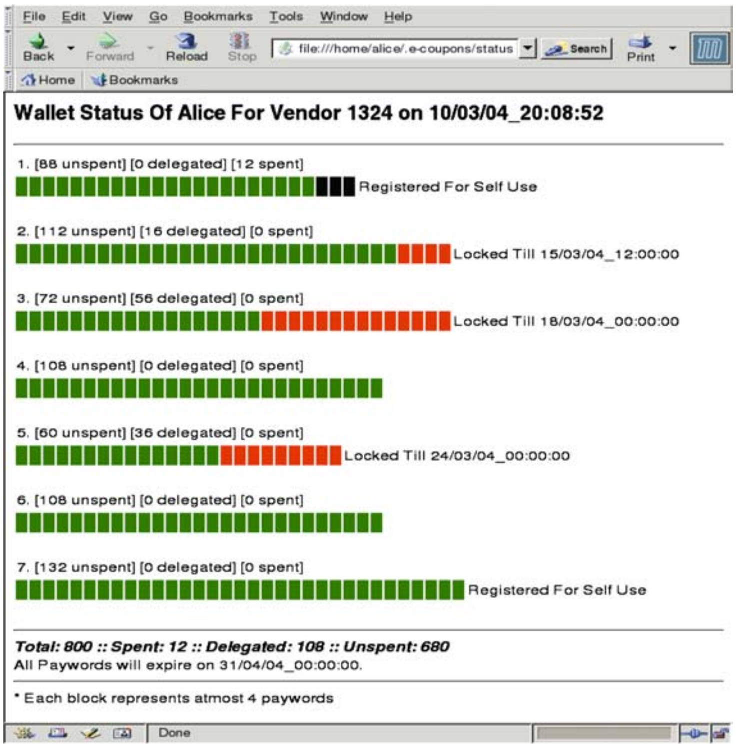

Fig. 7. *User's wallet status at time* t_1.

may have different methods to implement their interface for users. Vendor is equipped with SPKI/SDSI verification mechanism and a database to store user's information.

Users register with the vendor by sending the signed commitments along with their proof of spending capability either obtained from the facilitating bank or an authorized user. Upon receiving such

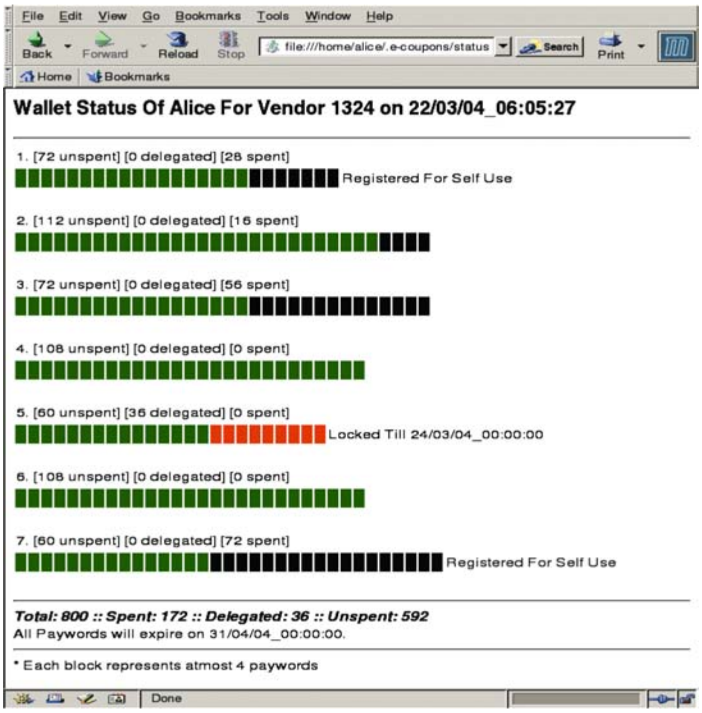

Fig. 8. *User's wallet status at time t_2.*

registration requests, Vendor performs the verification mechanism. The process involves user's credential (single PayWord certificate or chain of certificates—in case of delegation) verification and on successful verification Vendor stores the user's commitment in its database against the entry of user. Vendor retains this information for sufficiently long duration to thwart user's overspending and double-spending efforts. Before initializing the transaction, Vendor time-synchronizes itself with User by sending a request to User's port 2345. As the transaction proceeds, Vendor goes on storing the highest payword index received from the user.

Vendor periodically sends the accumulated commitments and payword indices to the facilitating Bank for redemption. Bank may report existence of double-spending efforts from particular user. Vendor immediately purges such users from the database of registered users.

6. e-coupons*: Applications*

In this section, we provide a typical application scenario of *e-coupons* system which cannot be envisaged using existing schemes. But the scope of *e-coupons* should not be considered limited to the exemplary scenarios discussed in this paper. The ability of *e-coupons* to support concurrent payments for a subscribed service from different service access points simultaneously is one of its important features. This feature is explained with the help of following example.

Google, a leading search engine, allows application programmers to access its search data through APIs (Application Program Interfaces). At present Google provides free access to its data via APIs with per user, per week access restrictions. May be due to lack of a suitable micro-payment scheme, Google is not switching to pay-mode. *e-coupons* is a perfect fit for this scenario.

Let us assume, a user "U1" subscribes to such API service with the help of a facilitating bank. The user writes a multi-threaded application, in which each thread individually requires the API's subscription service and capability to pay on its own. Because, large systems do load balancing between the servers therefore the API requests get randomly distributed among the system's servers [cf. Fig. 10] (hence single-seed payword chain is not suitable for such scenarios). One

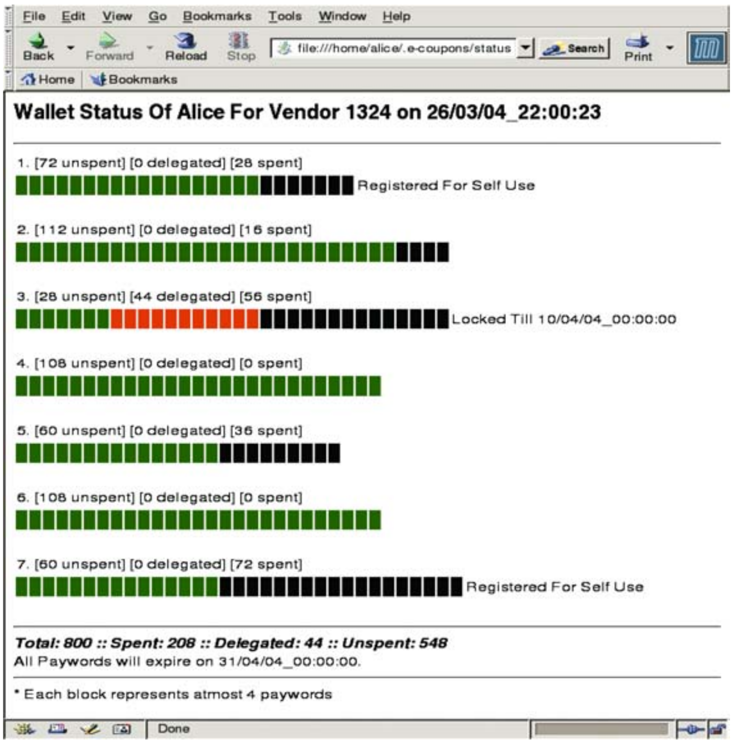

Fig. 9. *User's wallet status at time* t_3.

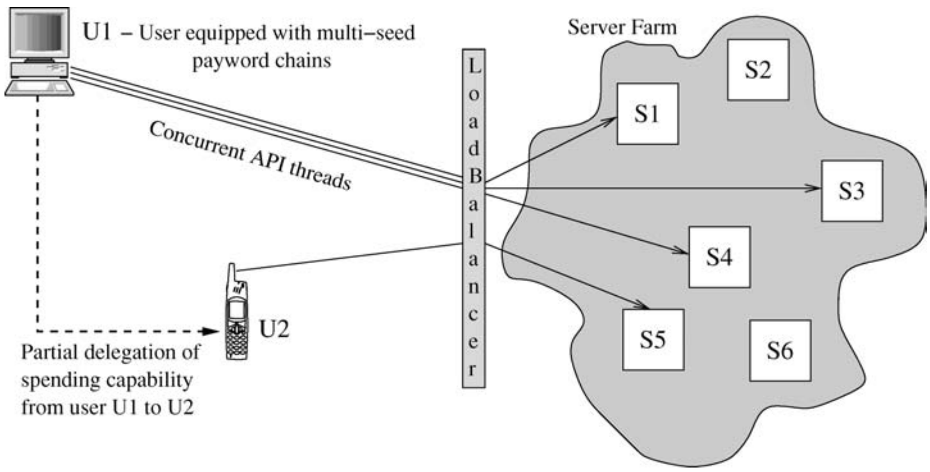

Fig. 10. *Concurrent payments for multi-threaded applications.*

may argue about making the "Load Balancer" itself as payment gateway for API service, instead of individual servers that are actually catering the API request. But such approach will centralize the computational burden to the "Load Balancer" and communication between the "Load Balancer" and the servers behind it will tremendously increase. One may also argue that instead of each thread making payments individually, let the parent of these threads make consolidated payments. But such an approach will limit the autonomy of the threads while negotiating for parameters like "Quality of Service".

Furthermore, a user would require to access the service from a temporarily borrowed device or may like to allow another user use the service in limited form (analogous to gift coupons). This requirement is depicted in Fig. 10, where user "U1" issues an authorization certificate to "U2". Apart from the validity period, this certificate contains the boundary values of a sub-chain, picked by "U1" from its un-locked set of multi-seed payword chains as shown in Section 4. The scenario described above has close resemblance with many applications in distributed computing, especially in Grid-Computing, pricing in MANETs etc. Furthermore, the facility of delegation enables the users to use the paywords fully, by gifting it to others, instead of them getting expired unspent.

7. *Analysis of* e-coupons *Scheme*

In this section, we provide analysis of *e-coupons* with respect to the issues like risk involved, security and performance, followed by a brief comparison of our system with existing micro-payment systems.

7.1. Risk analysis

e-coupons does not attract any additional risk while making the unit-wise payments. Though making the paywords free from user-specific-ness involves the risk of the paywords being stolen in transit, but adequate security via a relative encryption process and the TESLA authentication mechanism thwarts such attempts. A low-level risk is associated with all the parties involved in the protocol. Since bank gives credit facility to users and a guarantee to the vendors for redemption against paywords, the mischievous efforts of overspending by the user keeps the bank at risk. However the legal agreements between the bank and the user will be a deterrent. There is a risk of a user not receiving the goods for which he has paid for. The risk associated with the user is low because the payments are unit-wise, but the vendor is at great risk of losing his reputation.

We understand that the use of the same key for encryption purpose and for computing MAC might lead to cryptographic weaknesses of the protocol. But we are interested in providing confidentiality to the paywords for a brief time interval during their transit, which we do by using a 64-bit substring of the payword itself.

While the vendor loosely time synchronizes itself with the sender in TESLA protocol, it does not know the propagation delay of the time synchronization request packet, so it has to assume that the time synchronization error is Δ. To remain on safer side we take the full round-trip time of the packet. Even if vendor loses one of the valid incoming packet, it can own its value on successfully receiving the next packet because of the self-authenticating nature of the paywords in the chain. The vendor can always go from the highest payword value towards the *commitment* value. Given such facilities, the vendor is ready to take risk of losing intermediate packets due to network errors.

Also, the vendor needs to buffer packets during the disclosure delay before it can authenticate them. And at times due to heavy load on the vendor, it becomes risky to simply drop the packets when the resource (incoming buffer space) is fully utilized. The problem can be solved by keeping the onus of buffering the packets during disclosure delay on the users. Moreover, by enclosing hash values of future paywords in an earlier packet will help in authenticating data in later payword packets as soon as they arrive. Thus verification can be done in real-time.

The risk of double-spending paywords can be neutralized by two methods. Either the vendor should maintain a buffer of registered *commitment* values and check each new registration against this buffer or it can opt to verify each payword chain *commitment* value with the bank. The later option is *on-line* and it is costly. This will be a policy decision of the setup based on the agreement between bank and the vendor. We exercise the real-world reputation model to check the misconduct of the entities involved in the setup. A bad reputation due to non-delivery of goods on successful payments by the users would cost the vendor in terms of loss in business. And the bank will penalize the misbehaving users (double-spending efforts) by refusing to issue the PayWord certificate at the time of renewal of the subscription.

7.2. Security analysis

In PayWord protocol, the paywords are user and vendor-specific, so they don't bear the threat of being stolen while in transit unlike *e-coupons*, where the paywords are not user-specific. Every message in which the adversary might have interest is encrypted using low-bit encryption keys. Security is provided to every

Table 1. *Performance analysis and comparative evaluation*

	No. of Asymmetric key operations required signature/encryption		No. of Symmetric key operations required payword encryption/decryption		No. of Hashes required payword gen./verification, MAC computations	
	User	Vendor	User	Vendor	User	Vendor
Chain 1	1 + E	D	400	400	(402 × 2) + 400	404 + 400
Chain 2	1 + E	D	400	400	(404 × 2) + 400	408 + 400
Chain 3	1 + E	D	400	400	(408 × 2) + 400	416 + 400
Chain 4	1 + E	D	400	400	(416 × 2) + 400	432 + 400
Single Chain (1600 paywords)	1	1	0 no security required, since	0 coins are vendor and user-specific	1601 + 0	1601 + 0

payword sent by a user to the vendor because the paywords are not user-specific and are vulnerable to get stolen while in transit. We have provided the security with the help of TESLA's efficient source authentication mechanism and by simultaneously encrypting each payword with itself. TESLA has not become an overhead since we do not generate a separate *one-way* hash chain for MAC computations and make use of the readily available self-authenticating payword chain itself; since it is another chain of *one-way* hash values. The double-spending of paywords is not possible since the paywords are vendor-specific.

The user credit card information is sent under standard payment protocol, whereas the encryption method used for payword's confidentiality is relatively weak. Since 56-bit encryption provides satisfactorily enough confidence against the brute-force attacks for a time period that is enough for payword getting verified by the receiver, we avoid using the full length of payword as encryption key. Because of such a practical security cover for the payments, user can think of taking a risk of sending paywords of higher denomination.

7.3. Performance analysis and comparative evaluation

The performance evaluation of our system against the original PayWord protocol while making a deviation from the monolithic user-specific and vendor-specific payword chain to a multi-seed vendor-specific payment instrument is given in Table 1. In this table, D denotes the number of times the encryption of the $Nonce$ is done as a time synchronization request from a vendor to an user, and E denotes the encrypted response from the user to the vendor. Value of both D and E is equal to the total number of transaction initiation phases between them, as time synchronization is the first step in doing micro-transactions. The symmetric key operations (encryption/decryption using 64-bit DES) are involved in our implementation at the stage of sending the paywords and their verification at the receiving end.

This analysis is based on delegation of payword authority up to depth 1, i.e., the sub-users who have received the payword authorization have not delegated their authority any further, which will be the general case. Therefore, in Hash column of Table 1, the user's payword chain length is multiplied by 2; the original owner of the chain generates the values applying hash function h and delegates some range of this chain to the sub-user and the sub-user again computes the values between the range. Hence, the multiplication factor is $d + 1$, where d is delegation depth.

Also, the delegation of payword authority by a user to a sub-user adds a certificate into the authorization proof (X) of the sub-user. This will increase the authorization verification time of the vendor. Therefore, the depth of delegation and the time required for authorization verification by the vendor are linearly proportional. From our implementation, we have produced a graph of "number of paywords" against "verification time required by the Vendor" for PayWord and *e-coupons* protocol to substantiate our claims (Fig. 11). It is evident from the graph that *e-coupons* require more time, on Vendor's side, for validation as compared to PayWord protocol but it provides the flexibility to the users to manage their spending capability.

The hash operations performed by sub-user for payword generation is a re-computation of values earlier computed by the delegator. So, it will be a policy decision, whether to give computed values to the user or only the boundary values. If a user skips some paywords and trades a later one without trading

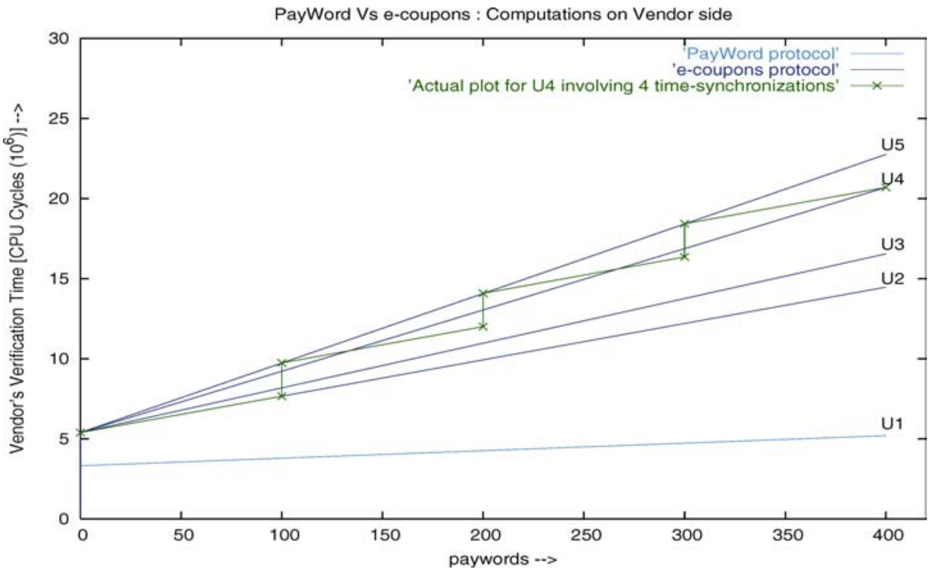

Fig. 11. *Computational evaluation of* e-coupons *against payword w.r.t. vendor*

.

those skipped ones, the user can pay a higher amount in one transaction. The vendor can still check the validity of the payword by a repeated application of the hash function. While implementing the measures against the double-spending, the process can be improved in its efficiency by employing probabilistic polling (Jarecki and Odlyzko, 1997). Now, let us evaluate *e-coupons* with the other existing schemes.

1. A micro-payment scheme requires PKI for authentication and non-repudiation. SPKI provides the primitive facilities required from a PKI. Because of this framework it is possible for us to introduce the concept of spending capability delegation to other users. This facility is absent in all other micro-payment schemes. Obviously, one cannot keep the paywords user-specific if one is going to delegate the authority to spend the paywords to others. This modification to the original PayWord scheme introduces the threat of paywords being snatched while in transit. We have shown how TESLA provides security to the paywords in transit. These gradual modifications to the original PayWord scheme makes our scheme slightly less efficient than the simple PayWord scheme. But the original PayWord scheme lacks the much required facility of delegation of the spending capability.
2. Amongst the various existing micro-payment schemes attempting to provide low-cost payments over the Internet, the most closely related to our scheme are PayWord and MiniPay. Both of them are *off-line* schemes and are suitable for *pay-as-you-go* applications. But our scheme differs from them substantially in terms of facilities like delegation and the way in which we provide security to the transactions.
3. Since PayWord scheme is vendor-specific and user-specific, it limits the user of a particular subscription to a single registered access point, which is quite un-natural. In practice, if a user is subscribed to a particular service, the user should have full freedom to access the service irrespective of the access methodology. Making the payword chains user-specific, the scheme has tackled the security and double-spending issues elegantly.
4. Unlike PayWord's tripartite architecture (consisting a bank, vendor and the users), MiniPay system would consist four to six parties (users, Access Provider for users' billing system, a bank, vendor, Internet Service Provider for seller's billing system, an arbitrator). Since trust is not transitive, the increase in number of entities in basic architecture for the sake of facilities like multiple currency support, overspending request handling etc., reduces the overall trust amongst the entities of the system. Instead the support for multiple currencies can be handled by the facilitating bank without substantial efforts on design side. We have used

currency neutral units for payment and the vendor can redeem such units from the bank in desired currency at current exchange rates between the currency in which user has paid for the PayWord Authorization and the currency desired by the vendor.

5. In MiniPay protocol, the threat of *denial-of-service* is not handled by the end users but by the intermediate facilitators i.e., Access Provider and the Internet Service Provider. It claims that all parties are protected from clogging. In contrast, the TESLA mechanism in our protocol allows the buyer and seller to do source authentication on each incoming packet in real-time. In this way, we have handled the bogus incoming packets intended for *denial-of-service* attack.

8. Conclusion

In this paper, we have discussed the design and implementation of an efficient micro-payment scheme that supports delegation of spending capability to others and has inherent lightweight security measures in it. To the best of our knowledge, this is the first of such initiative. It is a low cost, robust scheme and has negligible delay in response time. Our scheme detects attempts of double-spending, thwarts attempts of *denial-of-service* and *man-in-middle* attacks. The results of our implementation are satisfactory and show improved efficiency while integrated with the probabilistic signature verification scheme (Shamir, 1995).

In terms of trust associated among the three parties i.e., the bank, the users and the vendors and the risk involved in this protocol, our scheme is as good as PayWord scheme. Furthermore, PayWord assumes vendors to be *trusted*, while *users* need not be trusted. Our scheme also works under the same environment of trust and mistrust. However, our scheme gives more practical functionalities to the micro-payment transactions, like partial handing over of the spending capability to your application robots or offering introductory limited subscriptions to potential customers which are not part of such a setup. The introduction of delegation feature does not invite any risk, therefore the facility of delegation is viable.

The role of SPKI/SDSI is not restricted to delegation, but it also fulfills the system's requirement of a PKI providing public-keys and certificate validation, verification. Thus we have an excellent micro-payment system which is secure, relatively efficient, and provides one layer of indirection (while users delegate the authority) that contributes to transaction anonymity.

Our scheme allows a user to parameterize the security strength provided for the payments. For making payments at different vendors, user can specify the encryption strength to be provided for the payments based on the vulnerability of underlying protocol (HTTP, WAP) and available computational resources. This provision encourages a user to make payments of higher value with more security. A practically secure micro-payment with relatively higher monetary value is equivalent to making multiple efficient payments of the same monetary value for the same set of deliverables. Such provisions are important since they cater to transactions lying in between micro-payments and macro-payments.

Our implementation is *off-line*, vendor-specific but not user-specific and quite efficient. Being just vendor-specific, it is able to thwart the double-spending efforts and collusion between the vendors. Since it is not user-specific, it faces the risk of paywords getting stolen while in transit. We have done away with this problem by providing an *ephemeral* data confidentiality cover. The layers of indirection in authorizations give the end users more anonymity than they were enjoying earlier. Needless to say, privacy is a much demanded feature for e-commerce.

One can make the system free from vendor-specificness by making the vendors maintain the values of commitments they receive, and simultaneously (*on-line*) submitting it to the bank for verification against multiple registrations. This way, the user cannot spend the same payword chain with two different vendors taking the advantage of vendor's periodic settlement with the bank.

Appendix

A. One-way hash functions & MAC

A hash function is a mathematical function that takes a variable-length input string (called a pre-image) and converts it to a fixed-length (generally smaller) output string (called a hash value). A one-way hash function is a hash function that works in one direction:

```
(cert
 (issuer (name
  (hash sha1 |isEf64Sf5JpqasB4DCsR6Bn=|)
   Service-X-Subscriber-class-economy))
(subject
  (hash sha1 |Gf5tUySRhJiLk3syer0ibk3=|)
(not-before "2003-10-01_12:00:00")
(not-after "2004-10-31_11:59:59"))
```

```
(cert
 (issuer
  (hash sha1 |isEf64Sf5JpqasB4DCsR6Bn=|))
(subject (name
  (hash sha1 |isEf64Sf5JpqasB4DCsR6Bn=|))
   Service-X-Subscriber-class-economy))
(delegate) (tag (issue-payword
           (payword-limit 3000)
           (service http://xplore.ieee.org)))
(not-before "2003-10-01_12:00:00")
(not-after "2003-10-31_11:59:59"))
```

Fig. 12. *Name and authorization certificates (issued by the bank to a user).*

It is easy to compute a hash value from pre-image, but it is hard to generate a pre-image that hashes to a particular value. The output is not dependent on the input in any discernible way. A single bit change in the pre-image changes, on the average, half of the bits in the hash value. Given a hash value, it is computationally infeasible to find a pre-image that hashes to that value. A good one-way hash function is also collision-free: It is hard to generate two pre-images with the same hash value (Schneier, 1996; Lamport, 1981). So, a one-way hash function is a mapping h from some set of words into itself such that:

1. Given a word x, it is easy to compute $h(x)$.
2. Given a word y, it is not feasible to compute a word x such that $y = h(x)$.

A message authentication code, or MAC, is a key-dependent one-way hash function. MACs have the same properties as the one-way hash functions, but they also include a key. Only someone with the identical key can verify the hash (Schneier, 1996). They are very useful to provide authenticity without secrecy.

B. SPKI certificates

Fig. 12, shows the actual certificates issued by the bank B, whose public-key is marked inside the `issuer` box, to the user U as a `subject` of the certificate. By issuing this name certificate, Bank has given the subscriber a membership to its group `Service-X-Subscriber-class-economy` for a period of one-year. And by issuing the authorization certificate, the bank has empowered the members of group `Service-X-Subscriber-class-economy` to mint their coins with proper limits (in this case it is 3000, similarly there can be another group definition called `Service-X-Subscriber-class-exclusive` with higher spending limits). This way SPKI has clear edge over other PKI schemes in terms of efficient management of name space and authorization. Also, note the difference between the validity periods of the two certificates. The authorization expires within a month, whereas the name bindings stand for longer time-period.