Formal Specification and Verification of a Micropayment Protocol

Mohamed G. Gouda and Alex X. Liu*

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188, U.S.A.
{gouda, alex}@cs.utexas.edu

Abstract

In this paper, we investigate the security of micropayment protocols that support low-value transactions. We focus on one type of such protocols that are based on hash chains. We present a formal specification of a typical hash chain based micropayment protocol using Abstract Protocol notation, and discuss how an adversary can attack this protocol using message loss, modification, and replay. We use convergence theory to show that this protocol is secure against these attacks. The specification and verification techniques used in this paper can be applied to other micropayment protocols as well.

1. Introduction

As online businesses grow, there is an increasing demand for micropayment protocols that facilitate microcommerce, namely selling content and services for small amounts of money (possibly less than one cent per transaction), which cannot be handled efficiently by credit cards due to substantial per transaction fee and delay. Examples of such content and services are web pages and online games. Many micropayment protocols have been proposed for microcommerce, such as [1,3,6,7,10–12]. These protocols need to be regarded as "secure" before they can win the approval of customers and vendors alike. However, none of these protocols have been formally specified and verified.

In this paper, we address this issue by formally specifying a micropayment protocol, which is based on Lamport's idea of hash chains [9], using the Abstract Protocol notation presented in [4], and formally verifying the security of this protocol using the convergence theory in [5]. We choose such a hash chain based micropayment protocol because it is a typical micropayment protocol and the techniques that we use in specifying and verifying this protocol can be applied to other micropayment protocols as well. There are many hash chain based micropayment protocols, such as Anderson's et al. NetCard [1], Hauser's et al. Micro-iKP [6], Jutla and Yung's Paytree [7], Pedersen's Scheme [10], Rivest and Shamir's PayWord [11], and W3C's MPTP [12].

The rest of this paper is organized as follows. In Section 2, we present a brief introduction to Abstract Protocol notation, while in Section 3, we formally specify the hash chain based micropayment protocol using this notation. In Section 4, we give an introduction to the convergence theory. In Section 5, we verify the security of the protocol. We give conclusions in Section 6.

2. Abstract Protocol Notation

In this section, we give a brief introduction to the Abstract Protocol notation [4]. In this notation, each process in a protocol is defined by sets of constants, variables, parameters, and actions. For instance, in a protocol consisting of two processes p and q and two opposite-direction channels, one from p to q and one from q to p, process p can be defined as follows:

```
process p
const  ⟨name of constant⟩ : ⟨type of constant⟩
       ...
       ⟨name of constant⟩ : ⟨type of constant⟩
inp    ⟨name of input⟩    : ⟨type of input⟩
       ...
       ⟨name of input⟩    : ⟨type of input⟩
var    ⟨name of constant⟩ : ⟨type of constant⟩
       ...
       ⟨name of constant⟩ : ⟨type of constant⟩
par    ⟨name of constant⟩ : ⟨type of constant⟩
       ...
       ⟨name of constant⟩ : ⟨type of constant⟩
```

1 Alex X. Liu is the corresponding author of this paper.

```
begin
        ⟨action⟩
□       ⟨action⟩
□       · · ·
□       ⟨action⟩
end
```

The constants of process p have fixed values. Inputs of process p can be read, but not updated, by the actions of process p. Variables of process p can be both read and updated by the actions of process p. Comments can be added anywhere in process p; every comment is placed between the two brackets { and }.

Each ⟨action⟩ of process p is of the form:

$$\langle \textit{guard} \rangle \ \rightarrow \ \langle \textit{statement} \rangle$$

The guard of an action of process p is of one of the following three forms: (1) a boolean expression over the constants and variables of p, (2) a receive guard of the form "**rcv** ⟨*message*⟩ **from** q", (3) a timeout guard that contains a boolean expression over the constants and variables of every process and the contents of all channels in the protocol. A parameter declared in a process is used to write a finite set of actions as one action, with one action for each possible value of the parameter.

Executing an action consists of executing the statement of the action. Executing the actions of different processes in a protocol proceeds according to the following three rules. First, an action is executed only when its guard is true. Second, the actions in a protocol are executed one at a time. Third, an action whose guard is continuously true is eventually executed.

The ⟨*statement*⟩ of an action of process p is a sequence of ⟨*skip*⟩, ⟨*send*⟩, ⟨*assignment*⟩, ⟨*selection*⟩, or ⟨*iteration*⟩ statements of the following forms:

```
⟨skip⟩        : skip
⟨send⟩        : send ⟨message⟩ to q
⟨assignment⟩  : ⟨variable in p⟩ := ⟨expression⟩
⟨selection⟩   : if ⟨boolean expression⟩ → ⟨statement⟩
                · · ·
                □ ⟨boolean expression⟩ → ⟨statement⟩
                fi
⟨iteration⟩   : do ⟨boolean expression⟩ → ⟨statement⟩
                od
```

There are two channels between the two processes: one is from p to q, and the other is from q to p. Each message sent from p to q remains in the channel from p to q until it is eventually received by process q. Messages that reside simultaneously in a channel form a sequence and are received, one at a time, in the same order in which they were sent.

3. Formal Specification

There are many hash chain based micropayment protocols such as Rivest and Shamir's PayWord [11], W3C's MPTP [12], Hauser's et al. Micro-iKP [6], and Anderson's et al. NetCard [1]. The basic ideas of these protocols are quite similar. Here we take PayWord as an example to explain this type of protocols.

There are three types of parties in PayWord protocol: users, vendors, and banks. Each user has a private key, a public key, and a certificate. The certificate contains the bank's name, the user's name, the user's public key and the expiration date. Each vendor knows each bank's public key and therefore can verify each user's certificate. All parties, including users, vendors, and banks, know a one-way hash function h.

Each day, when a user u needs to pay a vendor v for the first time, u at first conjectures the most likely maximum number of "coins" that she might need to pay the vendor that day, which is denoted n. Second, u picks a random number, which is denoted $c[n]$. Third, u computes a hash chain $c[0], c[1], \cdots, c[n]$, where $c[i-1] = h(c[i])$ for each i $(1 \leq i \leq n)$. The integer $c[0]$ is called the *root* of the hash chain and the rest n integers from $c[1]$ to $c[n]$ then serve as n coins. Before making payments using these coins, u needs to send $c[0]$, u's signature of $c[0]$, and u's certificate to the vendor v in order for v to know $c[0]$, which enables v to verify the payment from u. After the above preparation steps, u starts to make payments to v. There are two types of payments: *fixed-size* payments and *variable-size* payments. In fixed-size payments, the i-th payment from u to v contains the integer $c[i]$. The vendor v verifies this payment by applying the hash function h to $c[i]$ and compare it with the coin $c[i-1]$ that v has received in the previous payment from u. In variable-size payments, the value of each payment varies from 1 to n. A payment from u to v contains a tuple $(c[i],\ m)$, where m is the value of this payment and the coins from $c[1]$ to $c[i-m]$ have been spent. This single payment is equivalent to m fixed-size payments from $c[i-m+1]$ to $c[i]$. The vendor v verifies this payment by applying the hash function h to $c[i]$ for m times and compares the resulting number with the coin $c[i-m]$ that v received in the previous payment from u.

The above PayWord protocol has two security holes. First, it is vulnerable to message loss attack. An attacker can discard a payment message from u and v. When this happens, u should retransmit the lost payment. However, due to the lack of an acknowledgement mechanism in the PayWord protocol, u does not know whether a payment was actually received by v or not.

Second, the PayWord protocol is vulnerable to message modification attack. An attacker can modify a valid variable-size payment message $(c[i], m)$ $(m \geq 2)$ to a different yet valid payment $(h(c[i]), m-1)$. When this happens, v should discard the modified payment. However, in PayWord protocol, such a modified payment is deemed as a valid one by v. Similar to the PayWord protocol, W3C's MPTP, Micro-iKP and NetCard also suffer from the above two vulnerabilities (Note that in Micro-iKP and NetCard, the effect of a message modification attack is the same as a message loss attack because these two micropayment protocols do not support variable-size payments.)

The above two security holes of the PayWord protocol were previously discovered in [8]. However, the solutions proposed in [8] are inefficient. In [8], to counter message loss attacks, a vendor is also required to compute and store a different hash chain c' of length n and each element $c'[i]$ is used to acknowledge the coin $c[i]$ received from a user u. This solution is inefficient because it is possible that u only spends a few coins with v although u computes a hash chain of length n. Therefore, v wastes both time and space in computing and storing a hash chain of length n because she only needs to send a few acknowledgements. The solution to counter message modification attacks in [8] fails to enable a vendor to detect whether a variable-size payment has been modified or not when she receives it.

Our solution to these two security holes are much more efficient than the solutions proposed in [8]. First, we use a securely salted one-way hash function $h(ss, \cdot)$, where ss is not known to attackers, instead of a normal one-way hash function $h(\cdot)$. Due to use of this securely salted hash function $h(ss, \cdot)$, an attacker cannot modify a valid variable-size payment message to a different yet valid payment message. Using this hash function $h(ss, \cdot)$, a user computes a hash chain of length n by $c[i-1] = h(ss, c[i])$ for each i $(1 \leq i \leq n)$. Second, for each payment $(c[i], m)$ received by a vendor, the vendor sends back an acknowledgement $h(c[i], ss)$ to the sender. Note that an attacker might know $h(ss, c[i])$ since $h(ss, c[i]) = c[i-1]$, but she cannot forge $h(c[i], ss)$.

Next, we present a hash chain based micropayment protocol that uses our above solutions to fix these two security holes. For simplicity and elaboration of the above two security fixes, we only present the protocol between a user u and a vendor v. We also assume there is a shared key sk between u and v, which can be achieved by public key cryptography. Each hash chain created by u has a sequence number that starts at 0. There are two phases in this hash chain based micropayment protocol: *request-reply* phase and *pay-ack* phase.

In request-reply phase, u first picks two random numbers for $c[n]$ and ss, then computes a hash chain $c[0], c[1], \cdots, c[n]$, where $c[i-1] = h(ss, c[i])$ for each i $(1 \leq i \leq n)$, and sends a request message $rqst(NCR(sk, (c[0]|seq|ss))$ to v. Here $NCR(sk, (c[0]|seq|ss)$ is the encrypted message $(c[0]|seq|ss)$ by the shared key sk, where "$|$" denotes concatenation. The variable "*seq*" is the sequence number of the current hash chain. When v receives the request message $rqst(NCR(sk, (c[0]|seq|ss))$ from u, she decrypts it using the shared key sk and checks whether "*seq*" is the one that she expects. If so, v sends reply message $rply(c[0])$ to u; otherwise v discards the request message. When u receives this reply message, u knows that v received the request message correctly and u starts sending payments.

In the pay-ack phase, u sends a payment message $pay(c[i],\ m)$ to v. The value of this payment $pay(c[i],\ m)$ is m coins. When v receives this payment message, she applies the hash function h to $c[i]$ for m times and then checks whether the result is the coin that v received previously from u. If so, then v sends acknowledgement message $ack(h(c[i], ss))$ back to u; otherwise v discards the payment message. When u receives the acknowledgement message, she knows that v received the payment message correctly and she continues to send other payments.

The time chart that shows the message flow of this hash chain based micropayment protocol is shown in Figure 1. The two processes u and v are specified in Figure 2 using the Abstract Protocol notation. Note that "*any*" represents an arbitrary number chosen by a human being, and "*random*" represents a random number generated by a computer. Here, $\#ch.u.v$ denotes the number of messages in the channel from u to v. We use NCR and DCR to denote encryption and decryption functions respectively.

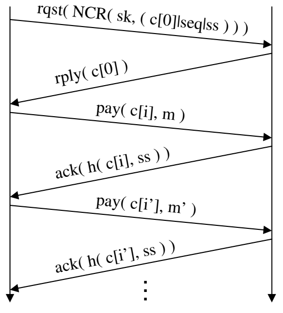

Figure 1. Time Chart

```
process u
const $sk$   : integer   {shared key between processes u and v}
var   $c$    : array integer of integer,   {current hash chain}
      $n$    : integer,  {length of current hash chain}
      $t$    : integer,  {c[t+1] is the next unspent coin}
      $seq$  : integer,  {sequence number of current hash chain}
      $ss$   : integer,  {session secret}
      $st$   : 0..3,     {state indicator whose initial value is 0}
      $m, x$: integer
begin
  $st = 0 \rightarrow$  $st := 1$;  $n :=$ any;  $t := n$;
                $c[n] :=$ random;  $ss :=$ random;
                do $(t > 0) \rightarrow c[t-1] := h(ss, c[t])$;  $t := t - 1$ od;
                send $rqst(NCR(sk, (c[0]|seq|ss)))$  to v
□ rcv $rply(x)$ from $v \rightarrow$
                if $st = 1 \wedge x = c[0] \rightarrow$  st:=2;  seq:=seq+1
                □ $st \neq 1 \vee x \neq c[0] \rightarrow$  skip
                fi
□ $st = 2 \rightarrow$  $st := 3$;  $m :=$ any;  $t := t + m$
                if $t \leq n \rightarrow$  send $pay(c[t],\ m)$  to v
                □ true $\rightarrow$  st:=0
                fi
□ rcv $ack(x)$ from $v \rightarrow$
                if $st = 3 \wedge x = h(c[t], ss) \rightarrow$  st:=2;
                □ $st \neq 3 \vee x \neq h(c[t], ss) \rightarrow$  skip
                fi
□ timeout $(\#ch.u.v + \#ch.v.u = 0) \wedge (st = 1 \vee st = 3)$
                if $st = 1 \rightarrow$  send $rqst(NCR(sk, (c[0]|seq|ss)))$  to v
                □ $st = 3 \rightarrow$  send $pay(c[t],\ m)$  to v
                □ $st \neq 1 \wedge st \neq 3 \rightarrow$skip
                fi
end
```

```
process v
const $sk$     : integer   {shared key between processes u and v}
var   $lastc$  : integer,  {last coin received}
      $seq$    : integer,  {index of sticks}
      $ss$     : integer,  {session secret}
      $lastc', seq', ss', x, c, m, j$ : integer,
begin
  rcv $rqst(x)$ from $u \rightarrow$
      $lastc', seq', ss' := DCR(sk, x)$;
      if $seq' = seq$      $\rightarrow$  $lastc = lastc'$;  $ss = ss'$;  $seq = seq + 1$;
                             send $rply(lastc)$ to u;
      □ $seq' = seq - 1 \rightarrow$  send $rply(lastc)$ to u;
      □ $seq' \neq seq \wedge seq' \neq seq - 1 \rightarrow$skip
      fi
□ rcv $pay(c, m)$  from $u \rightarrow$
      if $c \neq lastc \rightarrow$  $j := m$;  $x := c$;
                   do $(j > 0) \rightarrow x := h(ss, x)$;  $j := j - 1$ od;
                   if $x = lastc \rightarrow$  send $ack(h(c, ss))$ to u;
                                    $lastc := c$;
                   □ $x \neq lastc \rightarrow$  skip
                   fi
      □ $c = lastc \rightarrow$  send $ack(h(c, ss))$ to u;
      fi
end
```

Figure 2. Formal Specification

4. Convergence Theory

In this section, we outline a verification method, which is based on the three concepts from convergence theory [2, 5], namely closure, convergence, and protection, for verifying the security of protocols that are specified using the Abstract Protocol notation. Later in Section 5, we use this method to verify the security of the PayWord micropayment protocol. This verification method is based on the following definitions.

A *state* of a protocol is an assignment of a value to each variable of each process in the protocol and an assignment of a sequence of messages to each channel in the protocol. The value assigned to each variable is from the domain of that variable. If the guard of an action of a process in a protocol has the value true at some state of the protocol, then the action is said to be *enabled* at that state. For simplicity, we assume that at each state of a system, at least one action of that system is enabled.

Some states of a protocol are called the *initial states* of that protocol.

A *transition* of a protocol is a pair (p, q) of states of the protocol such that some process in the protocol has an action whose guard is true at state p and execution of that action when the protocol is at state p yields the protocol at state q.

A *computation* of a protocol is an infinite sequence $(p.0, p.1, p.2, \cdots)$ of protocol states such that each pair $(p.i, p.(i+1))$ of successive states in the sequence is a protocol transition.

A state of a protocol is called a *safe state* if it occurs in any protocol computation $(p.0, p.1, p.2, \cdots)$ where $p.0$ is an initial state of the protocol.

A state of a protocol is called an *error state* if the protocol can reach this state by an adversary executing one of its actions starting from a safe state of the protocol.

A state of a protocol that is not safe is called an *unsafe state* if it is an error state of the protocol or if it occurs in any protocol computation $(p.0, p.1, p.2, \cdots)$ where $p.0$ is an error state of the protocol.

A protocol is called secure if it satisfies the following three conditions:

1. **Closure**: In each protocol computation whose first state is safe, every state is safe.
2. **Convergence**: In each protocol computation whose first state is unsafe, there is a safe state.
3. **Protection**: In each protocol transition, whose first state is unsafe, the critical variables of the protocol do not change their values.

According to the above definitions, every protocol satisfies the closure condition. Thus, to prove that a protocol is secure, it is sufficient to show that the protocol satisfies both the convergence and protection conditions.

5. Formal Verification

In this section, we formally verify that the hash chain based micropayment protocol specified in Section 3 is secure against message loss, modification and replay attacks.

First, we examine the state transition diagram of this protocol without adversary actions, which is shown in Figure 3. The six safe states, $S.1$ through $S.6$, are defined in Figure 4. Here seq_u denotes the variable seq in process u, similarly for seq_v, ss_u, etc. Note that the action $u.i$ denotes the i-th action in process u and the action $v.i$ denotes the i-th action in process v.

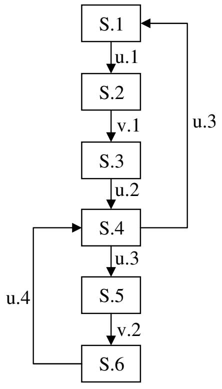

Figure 3. State Transition Diagram without Adversary Actions

$$
\begin{aligned}
&\text{S.1}: st = 0 \wedge seq_u = seq_v \wedge \#ch.u.v = 0 \wedge \#ch.v.u = 0\\
&\text{S.2}: st = 1 \wedge seq_u = seq_v\\
&\qquad \wedge ch.u.v = \{rqst(NCR(sk, (c[0] | seq_u | ss_u)))\} \wedge \#ch.v.u = 0\\
&\qquad \wedge (\forall i : 0 \leq i < n : c[i] = h(ss_u, c[i+1]))\\
&\text{S.3}: st = 1 \wedge seq_u = seq_v - 1 \wedge ss_u = ss_v \wedge lastc = c[0]\\
&\qquad \wedge \#ch.u.v = 0 \wedge ch.v.u = \{rply(c[0])\}\\
&\qquad \wedge (\forall i : 0 \leq i < n : c[i] = h(ss_u, c[i+1]))\\
&\text{S.4}: st = 2 \wedge seq_u = seq_v \wedge ss_u = ss_v \wedge \#ch.u.v = 0\\
&\qquad \wedge \#ch.v.u = 0 \wedge (\forall i : 0 \leq i < n : c[i] = h(ss_u, c[i+1]))\\
&\text{S.5}: st = 3 \wedge seq_u = seq_v \wedge ss_u = ss_v \wedge ch.u.v = \{pay(c[t],\ m)\}\\
&\qquad \wedge \#ch.v.u = 0 \wedge lastc = h^m(ss_v, c[t])\\
&\qquad \wedge (\forall i : 0 \leq i < n : c[i] = h(ss_u, c[i+1]))\\
&\text{S.6}: st = 3 \wedge seq_u = seq_v \wedge ss_u = ss_v \wedge lastc = c[t]\\
&\qquad \wedge \#ch.u.v = 0 \wedge ch.v.u = \{ack(h(c[t], ss_v))\}\\
&\qquad \wedge (\forall i : 0 \leq i < n : c[i] = h(ss_u, c[i+1]))
\end{aligned}
$$

Figure 4. States $S.1$ through $S.6$

Second, we examine the state transition diagram of this protocol with adversary actions. Figure 5 and 6 show the state transition diagrams of the request-reply phase and pay-ack phase of the protocol respectively with adversary actions. The adversary actions are labelled L (for message loss), M (for message modification), and R (for message replay). The additional states that result from the adversary actions, labelled $L.1$ through $L.6$, $M.1$ through $M.4$, and $R.1$ through $R.6$, are all unsafe states. Note that action T denotes the timeout action in process u.

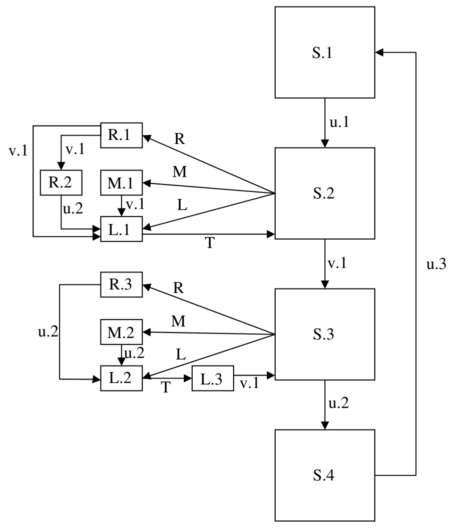

Figure 5. State Transition Diagram of Request-Reply Phase with Adversary Actions

As an example, we examine the four unsafe states (namely $L.1$, $M.1$, $R.1$, and $R.2$) that result from adversary actions when the protocol is in state $S.2$. In state $S.2$, there is one request message in the channel from u to v. If an adversary launches a message loss attack, i.e., discards the request message from u to v, then the protocol moves to an unsafe state $L.1$. In state $L.1$, only the timeout action T of process u is enabled and eventually executed, and henceforth brings the protocol back to the safe state $S.2$. If an adversary launches a message modification attack, i.e., modifies the request message in the channel from u to v, the protocol moves to unsafe state $M.1$. In state $M.1$, only action $v.1$ is enabled and eventually executed. In action $v.1$, v decrypts the message using the shared key sk. By checking the sequence number in the request message, v detects that the modified message is not a valid one, and therefore discards the modified message, which brings the protocol back to the unsafe state $L.1$. If an adversary launches a message replay attack, i.e., replaces the original request message with one of previous request messages, the protocol moves to an unsafe state $R.1$. There are two possible cases for the replayed message. If it is the most recent request message from u to v, then v sends back to u the most recent reply message. In this case, the protocol moves to another unsafe state $R.2$. But when u receives this reply message, u discards it because $st \neq 1$, which moves the protocol to state $L.1$. If the replayed message is not the most recent request message from u to v, v discards the message and henceforth moves the protocol to state $L.1$.

Figure 6 shows the state transition diagram of the pay-ack phase of the protocol with adversary actions. From Figure 5 and 6, we conclude that the protocol does satisfy the convergence condition because any computation whose first state is an unsafe state has a safe state.

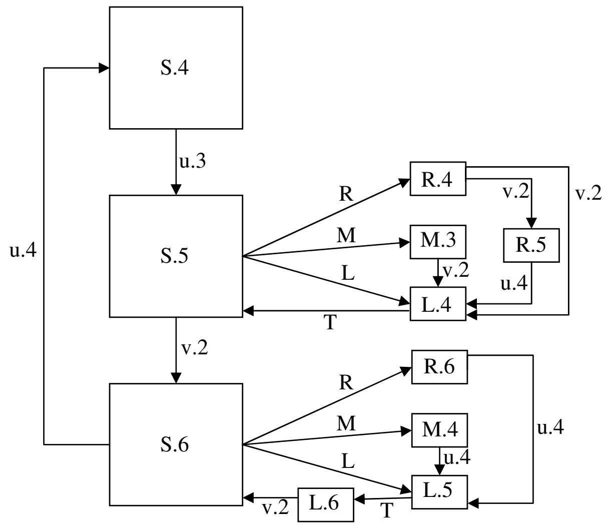

Figure 6. State Transition Diagram of Pay-Ack Phase with Adversary Actions

Next we prove that the protocol satisfies the protection condition by showing that no critical variables are updated when the protocol is in an unsafe state. The protocol has nine critical variables, namely c, n, t, seq, ss and st of process u, and $lastc$, seq and ss of process v. By examining the unsafe states in Figure 5 and 6, we see that any of the above nine critical variables is updated only in safe states. For example, the variable $lastc$ in process v is updated only when v receives a valid request message or a valid payment message.

In conclusion, the hash chain based micropayment protocol is secure against message loss, modification and replay attacks.

6. Conclusions

Our contributions in this paper are three-fold. First, we present two security fixes to the previous hash chain based micropayment protocols. Second, we formally specify a hash-chain based micropayment protocol using Abstract Protocol notation. Third, we formally verify that this protocol is secure against message loss, modification, and replay attacks using convergence theory. The specification and verification techniques used in this paper can be applied to other micropayment protocols as well.

References

[1] R. Anderson, H. Manifavas, and C. Sutherland. Netcard: A practical electronic cash system. In *Proc. of the 4th International Workshop on Security Protocols, LNCS 1189*, 1996.

[2] A. Arora and M. G. Gouda. Closure and convergence: A foundation for fault-tolerant computing. *IEEE Transactions on Software Engineering, Special Issue on Software Reliability*, 19(3):1015–1027, 1993.

[3] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro. The millicent protocol for inexpensive electronic commerce. In *Proc. of the 4th International World Wide Web Conference Proceedings*, pages 603–618, 1995.

[4] M. G. Gouda. *Elements of Network Protocol Design.* John Wiley & Sons, New York, New York, 1th edition, 1998.

[5] M. G. Gouda. Elements of security: Closure, convergence, and protection. *Information Processing Letters*, 77:109–114, 2001.

[6] R. Hauser, M. Steiner, and M. Waidner. Micropayments based on ikp. In *Worldwide Congress on Computer and Communications Security Protocol*, 1996.

[7] C. Jutla and M. Yung. Paytree: "amortised-signature" for flexible micropayments. In *Proc. of the 2nd USENIX Association Workshop on Electronic Commerce*, pages 213–221, 1996.

[8] A. Lakhia. Specification and verification of payword protocols. Bachelor's thesis, The University of Texas at Austin, 1998.

[9] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–771, 1981.

[10] T. P. Pedersen. Electronic payments of small amounts. In *Proc. of the 5th International Workshop on Security Protocols, LNCS 1361*, pages 59–68, 1997.

[11] R. L. Rivest and A. Shamir. Payword and micromint: Two simple micropayment schemes. In *Proc. of the Fourth International Workshop on Security Protocols, LNCS 1189*, pages 69–87, 1996.

[12] W3C. Micro payment transfer protocol (mptp) version 0.1. http://www.w3.org/tr/wd-mptp-951122.