SNPP: A Simple Network Payment Protocol

by

Semyon Dukach
B.S. Computer Science
Columbia University
(1990)

Submitted in Partial Fulfillment of the Requirements of the Degree of

Master of Science
in
Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology
October 1992

Signature of Author ______________________________

Department of Electrical Engineering and Computer Science
October 7, 1992

Certified by ______________________________

Karen R. Sollins
Thesis Supervisor

Accepted by ______________________________

Campbell L. Searle
Chairman, Departmental Committee on Graduate Students

SNPP: A Simple Network Payment Protocol

by

Semyon Dukach

Submitted to the
Department of Electrical Engineering and Computer Science
on August 14, 1992
in partial fulfillment of the requirements for the Degree of
Master of Science in Electrical Engineering and Computer Science

Abstract.
A protocol is proposed to securely implement payment transactions between mutually distrustful parties. This protocol is designed to operate over an open network, and can be implemented using currently available encryption technology.

Thesis Supervisor:
Karen R. Sollins
Research Scientist, Laboratory for Computer Science

Acknowledgements

My greatest gratitude goes to Galit P.G. Dukach, my wife and soulmate, without whom I'd be nowhere at all.

I would like to thank my former advisor, David D. Clark, for helping me find a topic to titillate my fickle interests, and my final advisor, Karen R. Sollins for always being there to guide me through my efforts.

I also wish to thank the following individuals, each of whom made a significant contribution to this endeavor, through technical assistance, motivation, or inspiration: Ken Hechtman, Jim Morrison, Vladimir Visotski, J. Laurens Troost, Ethan Rafferty, Adam Kucznetsov, Alex Peshkovski, Christopher Grewe, David Wetherall, Scott Susin, David Eichberg, Ray Hirschfeld, John Chang, Mitch Charity, Michele Rizack, Anna Bronstein, Magdalena Leuca, Inna Dukach, K.J. Madera, Leonid Fridman, Hamish Fletcher, Dina Dukach, David Dukach, David Tennenhouse, Petros Butler, Nahdiah Wiggins, Bob Jonas, Harris Gilliam, Arkadi Pershman, Greg Troxel, Henry Houh, Joseph Elkies, Alan Bawden, J.P. Massar, Derek Williamson, Kim Carter, Tim Shepard, Chuck Davin, Chris Lindblad, Ayn Rand.

1 Introduction

When the Internet was first created, many exciting services and applications were envisioned. It was widely believed that the network would be used to fulfill diverse needs and to provide many new capabilities for users around the world. Powerful services providing news, shopping, entertainment, and remote computation and storage were believed to be around the corner. Yet despite the fact that today's Internet has grown to sufficient scope and speed, very few advanced services are available at the moment. The large majority of traffic still consists of simple electronic mail and "news" delivery, remote logins, and file transfers.

We believe that a major reason why advanced applications have not flourished is the lack of a practical payment facility. Without the ability to efficiently exchange money, there is insufficient motivation for the development of production-quality software. In addition, since further growth of the Internet seems much less likely to rely on government funding, payment mechanisms are also needed to enable link owners to efficiently charge their users.

Informal *ad hoc* financial transactions have already begun to occur on the Internet. Most of these, such as sending credit card numbers through e-mail, are extremely insecure and inefficient. It is instead desirable to allow individuals with access to the Internet to pay for goods and services directly.

This thesis describes SNPP, a simple payment protocol for secure transactions over an open network. A logical verification of the protocol is included, as well as a status report on its implementation.

2 Payment Paradigms

The methods of payment in wide use today are cash, credit cards, direct billing, and checks. All four can potentially be implemented in some form on the Internet.

The major advantage of cash transactions is the anonymity of the buyer. Once the money changes hands, no one can discover where it has been. There are several cryptographic approaches to designing an electronic form of cash, a few of which have already been implemented with some success [5]. The main disadvantage that they all share is the relative complexity of the protocols.

A straightforward credit card model can be implemented on the network securely using some of the authentication and authorization protocols mentioned in section 4. But this model contains one undesirable assumption: in order to be able to receive credit cards payments, one must be officially approved as a merchant, and in a sense be trusted by the credit card company. This approval is generally only granted to established businesses, since the security of the payment protocols relies on the merchant not tampering with the point-of-sale terminal. It would be better to have a ubiquitous method of payment which anyone with network access can use in order to make *or* receive payments with a minimum of overhead formalities.

Direct billing is the most common way available today to pay for information. The problem is that it is impractical to provide granularity by billing for small amounts of money. For example, someone who might only need an occasional datum must subscribe to the entire service on a monthly basis.

The existing payment paradigm which most closely resembles the protocol proposed in this thesis is personal checks. Anyone can write checks after simply opening a bank account, and anyone who can write them can also receive them without additional registration. An electronic check can be cleared with a bank before a product is shipped, but actual payment can be delayed until the product is received. And since a check will be represented by a short sequence of data packets, a separate payment can be made for a series of small transactions.

3 Goals

Our payment protocol has been designed to meet the following goals:

- It should provide a secure means for payment transactions between parties over an open network.
- The parties should not need to trust each other, only their respective banks.
- A thief with the ability to arbitrarily view, store, and replace messages among all parties should at most cause denial of service, and should not be able to embezzle funds, even if allied with one of the parties.
- A mechanism should be provided for "holding" the funds during the sensitive period of a transaction, so that no money is handed over until a product is received, yet no product is sent out until payment is assured.
- The protocol should be fully partitionable, in the sense that a customer should only deal with his merchant, and a merchant only with her

bank. Therefore at each point, further verification could be postponed for batch processing at the risk of the appropriate party, and without a separate procedure at the beginning of the authorization chain.

For our initial implementation, the basic requirements are library calls providing UDP datagram communication, and DES encryption. In addition, several support functions such as reliable key distribution, stable storage for used transaction numbers, and workstation authentication will be necessary for an implementation secure enough for experimenting with non-trivial amounts of real currency.

4 Background

In order to provide the ability to transfer payments in real time, a mechanism is necessary for secure communication. In 1978, Needham and Schroeder [8] defined protocols for establishing secure interactive connections by using an authentication server. The Kerberos [12] authentication system extends the Needham and Schroeder algorithm to multiple authentication servers. Authorization and accounting schemes can be built on top of the authentication interface that Kerberos provides. The latest version of Kerberos also supports an extension defined by Neuman [10] as *Restricted Proxies.* The recipient of such a proxy is allowed to act with the same rights and privileges as the grantor of the proxy, subject to an arbitrary set of restrictions. Neuman briefly outlines how a payment protocol can be implemented on top of restricted proxies.

While restricted proxies and Kerberos may prove to be a desirable platform in future implementations, the protocol we are currently proposing has been designed from scratch, using UDP for communication, and DES for encryption. This has enabled us to give closer consideration to the fundamental security and performance tradeoffs inherent in the problem of open network payments. In addition, direct implementations allow the inclusion of several desirable properties, such as anonymity between the buyer and the seller, and partitionability of payment verification.

A network payment protocol is only as secure as the end workstations. In particular, a user has to be able to verify that the workstation is not hostile before divulging his key. As shown in Abadi *et al* [1], workstation authentication can only be achieved with a smart card which minimally possesses a self-powered clock, an internal encryption and decryption capability, and either a keyboard or a display. In their 1990 report, Bos and Chaum [2] describe an electronic payment system which employs smart cards capable of

DES but not RSA encryption to authenticate host workstations. Since our goal has been to design a payment system for the immediate future which does not require any special hardware, the protocol assumes that the user's end workstation is secure, without making any assumptions about the security of the network.

The proposed protocol will ultimately rely on an adequate solution to the problems of key distribution and storage. Although a lot of promising cryptographic work has been done in these areas, [11, 4, 7] SNPP will initially work fine if the keys are simply distributed physically, and stored in a DES encrypted file on a unix workstation which has most services turned off.

In order to verify the correctness of our protocol, we make use of *A Logic of Authentication*, developed by Burrows, Abadi, and Needham [3]. The proof of the correctness of SNPP outlined below is analogous to the verification of Kerberos given as an example in their paper. As demonstrated by Nessett [9], *BAN* logic cannot always be safely applied to the privacy aspects of protocols such as SNPP. Nonetheless, we found *BAN* verification, and especially the process of generating the "idealized protocol," to be very useful.

5 Summary of the Protocol

In order to act as either a customer or a merchant, a user must first open an account with a trusted bank and deposit some funds. At the time an account is opened, a key is generated for subsequent symmetric encryption, and is given to the user along with an account number. The bank keeps a record of each account, indexed by account number. The record contains the amount and type of currency in the account, a list of holds on the currency, the aforementioned key, the most recent transaction number, and a list of all transaction numbers smaller than the most recent one which have not yet been used. Each hold consists of an amount, an account number/bank id pair identifying the account for which the money is being held, and a timeout. A single person can maintain multiple accounts, and each account can be used for buying as well as for selling.

People who have accounts in different banks can still make payments to each other, subject to the condition that their banks trust each other, and have a pre-arranged common key.

The protocol begins when a customer issues a **HOLD** message to the merchant, which the merchant in turn forwards to her bank. The merchant's bank then forwards the **HOLD** to the customer's bank, if necessary. If the funds are available, the customer's bank adds the hold to the holds list of the customer's account, and returns a confirmation to the merchant's bank. After the merchant gets notified of the hold by her bank, she sends out the product to the customer. At this point, the customer sends out a **PAY** message to the merchant, and an analogous procedure occurs.

6 Notation

The following notation is used in the detailed presentation of the protocol in section 7:

- C — Customer.
- M — Merchant.
- B_i — Bank of person i.
- A_i — Account number of person i, changed periodically. A_i need not be confidential as it grants no privileges without K_i.
- K_i — The confidential key corresponding to an account of person i, changed periodically. K_i is also known by B_i given A_i.
- K_{B_i,B_j} — The common key between bank i and bank j.

- N_i — Transaction number assigned by person i (N_i is a nonce which is incremented after each complete transaction).
- P — The product, quantity and price information.
- \$ — The type and amount of currency.
- $\{x\}_{K_i}$ — Message consisting of x encrypted in the key K_i.
- $i \Longrightarrow j : x$ — i sends a message consisting of x over the net to j. A hostile party may tamper with x in any way.

This notation should not be taken to imply that there can only be one bank account per person. Further subscripting could clarify this point, but has been omitted for simplicity.

7 The Protocol

Before the beginning of the payment protocol, the merchant will probably advertise the product over a mailing list, a newsgroup, or some alternative method of distribution. The ad will include the bank id and account number to which the merchant expects the customer to direct his payments. This does not necessarily disclose any sensitive information about the merchant, since multiple accounts may be maintained with each bank, and account numbers can be automatically changed periodically. Once a mutually acceptable price is established, the protocol proceeds as follows:

1.

$$C \Longrightarrow M : P, B_C, A_C, \{B_M, A_M, N_C, \$, \textbf{HOLD}\}_{K_C}$$

The **HOLD** symbol within the encrypted portion of the message signifies that the money is to be held by the bank, and not actually transferred. The transaction number is needed to keep a thief from storing messages, and then replaying them again. Since only the customer's bank knows K_C, no one along the way can modify the sensitive fields, such as the amount of money involved.

2.

$$M \Longrightarrow B_M : B_C, A_C, \{B_M, A_M, N_C, \$, \textbf{HOLD}\}_{K_C}$$

The first two fields need not be encrypted, since forging them would simply prevent subsequent decryption of the **HOLD**, and would eventually set off alarms. The specific order information encapsulated in P is not passed on to the bank, since it is not needed.

3. If $B_M \neq B_C$ then

$$B_M \Longrightarrow B_C : A_C, \{B_M, A_M, N_C, \$, \textbf{HOLD}\}_{K_C}$$

If $B_M = B_C$ then messages 3 and 4 are of course unnecessary.

4. B_C now uses A_C to find K_C and decrypt the message. If N_C is greater than the previous N_C (or if it is on the list of unused skipped transaction numbers) and if the funds in A_C are available, B_C places a $\{B_M, A_M\}$ hold on the requested amount. The hold contains a standard timeout, after which it is automatically released. The following message is then generated: (If there is not enough currency in the account, replace **HELD** by **INSUFFICIENT FUNDS**, and increment a security alert counter).

$$B_C \Longrightarrow B_M : \{A_C, A_M, N_{B_C}, \$, \mathbf{HELD}\}_{K_{B_C, B_M}}$$

5. B_M decrypts the message using the inter-bank key of the bank implied by the return address. Then A_M is used to look up K_M and the address of M. Alternatively, the merchant's bank could remember the return address of the merchant's request by adding an extra reference field to message 3 encrypted in the inter-bank key. This would allow a merchant to initiate transactions from arbitrary addresses, but is omitted here for simplicity.

$$B_M \Longrightarrow M : \{B_C, A_C, N_{B_M}, \$, \mathbf{HELD}\}_{K_M}$$

The message is now encrypted in the merchant's key, so that the merchant can verify that the **HELD** guarantee is legitimate. When the merchant is able to decrypt this message, she will know that the customer must have supplied the correct A_M in the hold request. The reason that a new transaction number, N_{B_M}, is used here instead of N_{B_C}, is so that a merchant only has to keep track of transaction numbers emanating from one source: her bank. A_C and B_C are included in this message so that the **HELD** guarantee can be matched to the correct **HOLD** request.

6. At this point, M delivers the goods or services to C. If C is satisfied,

$$C \Longrightarrow M : P, B_C, A_C, \{B_M, A_M, N_C, \$, \mathbf{PAY}\}_{K_C}$$

7.

$$M \Longrightarrow B_M : B_C, A_C, \{B_M, A_M, N_C, \$, \mathbf{PAY}\}_{K_C}$$

8. If $B_M \neq B_C$ then

$$B_M \Longrightarrow B_C : A_C, \{B_M, A_M, N_C, \$, \mathbf{PAY}\}_{K_C}$$

The money is first taken from held funds, which are automatically released as they are paid out. If the **PAY** is more than the **HOLD**, then the rest is taken straight out of the account, if available.

9. B_C initiates an out-of-band procedure to transfer the funds to B_M.

$$B_C \Longrightarrow B_M : \{A_C, A_M, N_{B_C}, \$, \mathbf{IOU}\}_{K_{B_C,B_M}}$$

10. B_M transfers the requested amount into account A_M.

$$B_M \Longrightarrow M : \{B_C, A_C, N_{B_M}, \$, \mathbf{PAID}\}_{K_M}$$

If, after step 6, the customer does not receive the promised product, or is not satisfied with the quality, the **PAY** message is never generated, and the money remains on hold for the length of the timeout period, which should be chosen to be of sufficient length to allow for any possible disputes to be resolved. Likewise, if the merchant sends out the product as promised, but never receives the **PAID** message with the correct amount back from her bank, the merchant has not been fully paid, but all the money still remains held. In such cases, all claims to held funds will be addressed by an out-of-band arbitration procedure. If one of the parties cannot be contacted, the money would of course be awarded to the other; and otherwise, some sort of legal arbitration would occur.

There are other useful messages which should be provided with most implementations but have not been included here since they are not part of the core protocol. Some examples are messages that a merchant can send to release held funds in excess of the final payment before the timeout expires, messages account holders can send to their banks to receive account information, and so forth.

In addition to such steps, all users can automatically send out **UPDATE REQUEST** packets to their banks at regular intervals. Upon receiving such a packet, the bank generates a random unused account number and a random key, encrypts them with the user's old key, and returns them to the user. From then on until the next update request, communication will proceed using the new account number and key, although the old account number may continue to be referenced as long as there are outstanding holds for that account. Updating the key, of course, improves overall security, but updating the account number serves another function as well: since every customer will have a different account number every few transactions, merchants or network watchers would not be able to compile and sell lists of correlations between account numbers and purchases.

If the customer trusts that the merchant will deliver the product as promised, then the first half of the protocol may be skipped; that is, the customer may begin by sending out a **PAY** instead of a **HOLD** message.

Alternatively, if the merchant wants to save the cost of sending many small packets over the net, the product could be sent out immediately after step 1. It would be advisable, however, to proceed with the protocol to verify the hold for any customer originating from an unfamiliar address. In the same manner, the merchant's bank could issue the **HELD** guarantee message back to the merchant without bothering to contact the customer's bank. This would be more likely if accounts contain credit limits rather than deposited funds, and the banks use a common method for verifying credit worthiness and risk.

This partitionability of the protocol is the reason that per-party transaction numbers, rather than time, must be used to prevent replay attacks. If the merchant chooses to verify every Nth request from a particular customer, and processes the rest together at a later time, then the customer's bank will receive message N before messages 1 through $N-1$, so that merely comparing times would not be sufficient.

The main problem with the cost-saving scheme outlined in the above paragraphs is that on an open network, it is impossible to trace return addresses to individuals, and thus a thief could attempt numerous fake transactions without fear of retribution. One rationale for banks and/or merchants to skip the real-time verification step is the empirical expectation of losses, which would probably be bounded due primarily to the fact that most of the Internet maintains *some* level of security.

The use of transaction numbers within all encrypted messages should eliminate the risks associated with replay attacks. However, the possibility of inadvertent exposure of keys poses serious concerns. If a customer's or merchant's key somehow gets stolen, all the funds in the associated account are at risk of being embezzled. Furthermore, should an unauthorized party come into possession of an inter-bank key, all of the accounts at the two banks face potential danger.

8 Verification

To verify the correctness of SNPP we will employ the following constructs from the *BAN* Logic of Authentication [3]:

- P **believes** X — P believes that X is true, and is therefore free to act upon this belief.
- P **sees** X — P received a message containing X.
- P **said** X — P has sent a message containing X at some point in the past.

- **fresh**(X) — X has not appeared in a message processed in any previous run of the protocol; *i.e.,* X is a nonce.

- P **controls** X — P has the authority to determine X. For example, a customer has the authority to issue hold and pay requests from his account, and his bank respects that authority.

- $P \overset{K}{\leftrightarrow} Q$ — K is the shared key between P and Q.

- $\{X\}_K$ — Message X encrypted with the key K.

We will also make use of the following postulates from the logic:

- The *message-meaning* rule for shared keys:

$$\frac{P \textbf{ believes } Q \overset{K}{\leftrightarrow} P,\ P \textbf{ sees } \{X\}_K}{P \textbf{ believes } Q \textbf{ said } X}$$

  That is, if P sees a message X encrypted in a key that P shares with Q, then P believes that Q has said X at some point. The logic assumes that the message was not generated by P itself.

- The *nonce-verification* rule:

$$\frac{P \textbf{ believes fresh}(X),\ P \textbf{ believes } Q \textbf{ said } X}{P \textbf{ believes } Q \textbf{ believes } X}$$

  That is, if P believes that Q has said X at some point, but also believes that X has never been said in the past, then P must believe that Q

has said X in the present run of the protocol, and that at this point, Q **believes** X.

- The *jurisdiction* rule:

$$\frac{P \textbf{ believes } Q \textbf{ controls } X,\ P \textbf{ believes } Q \textbf{ believes } X}{P \textbf{ believes } X}$$

That is, if P believes that Q is an authority on X, and that Q believes X, then P can believe X also.

In order to verify SNPP, we must first convert it into the *BAN* idealized protocol form. Information sent in the clear is omitted, and all the information that a customer needs to deliver to his bank in order to place a particular hold or payment is encapsulated for the purpose of this verification in a **HOLDMSG** or **PAYMSG** variable.

1. $C \Longrightarrow M$: $\{N_C, \textbf{HOLDMSG}\}_{K_C}$

2. $M \Longrightarrow B_M$: $\{N_C, \textbf{HOLDMSG}\}_{K_C}$

3. $B_M \Longrightarrow B_C$: $\{N_C, \textbf{HOLDMSG}\}_{K_C}$

4. $B_C \Longrightarrow B_M$: $\{N_{B_C}, \textbf{HELDMSG1}\}_{K_{B_C,B_M}}$

5. $B_M \Longrightarrow M$: $\{N_{B_M}, \textbf{HELDMSG2}\}_{K_M}$

The second five messages are identical to the first, except the hold request is replaced by the payment request. The proof is analogous to the one below, and will not be given here.

The next step in analyzing the protocol is listing the assumptions. The following statements are assumed to hold when the protocol commences:

1. B_C **believes** $C \stackrel{K_C}{\leftrightarrow} B_C$

2. B_C **believes fresh**(N_C)

3. B_C **believes** (C **controls** hold-requests)

4. B_M **believes** $B_C \stackrel{K_{B_C,B_M}}{\longleftrightarrow} B_M$

5. B_M **believes fresh**(N_{B_C})

6. B_M **believes** (B_C **controls** held-messages)

7. M **believes** $B_M \stackrel{K_M}{\leftrightarrow} M$

8. M **believes fresh**(N_{B_M})

9. M **believes** (B_M **controls** held-messages)

In their idealized form, the first three messages simply pass the hold request along from C to B_C. When B_C receives message 3, the following statement holds, according to the definition of "**sees**":

$$B_C \textbf{ sees } \{N_C, \textbf{ HOLDMSG}\}_{K_C}$$

Since we have the assumption that B_C **believes** $C \overset{K_C}{\leftrightarrow} B_C$, the *message-meaning* rule applies, yielding

$$B_C \textbf{ believes } (C \textbf{ said } (N_C, \textbf{ HOLDMSG}))$$

Since we have assumed that B_C **believes fresh**(N_C), the *nonce-verification* rule applies, and yields

$$B_C \textbf{ believes } (C \textbf{ believes } (N_C, \textbf{ HOLDMSG}))$$

Breaking a conjunction,

$$B_C \textbf{ believes } (C \textbf{ believes HOLDMSG})$$

Since we have assumed that B_C **believes** that C **controls** hold-requests, and since **HOLDMSG** is an instance thereof,

$$B_C \textbf{ believes } (C \textbf{ controls HOLDMSG})$$

The *jurisdiction* rule now applies, yielding

$$B_C \text{ **believes HOLDMSG**}$$

Since B_C now believes the hold request, it will execute it, provided conditions such as fund availability are met.

After message 4 is received by B_M, we again apply the *message-meaning, nonce-verification, and jurisdiction* rules just like above, yielding

$$B_M \textbf{ believes HELDMSG1}$$

Now B_M can send out message 5 to the merchant, thereby telling her that it believes in the validity of the hold. When M receives message 5, the same three rules can again be applied to yield

$$M \textbf{ believes HELDMSG2}$$

Since in the concrete protocol the **HELD** message received by the merchant from her bank contains the amount that the customer has put on hold as well as information indicating that the hold is for an account belonging to this particular merchant, the merchant now has sufficient confidence in the hold to send out the product.

The proof of the payment half of the protocol is identical.

9 Implementation Status

A prototype implementation of SNPP is currently available for anonymous FTP from allspice.lcs.mit.edu. The implementation consists of the SNPP library, the customer, merchant, and bank programs, and various utilities. The most important functions provided by the library are the ones that process each of the protocol messages. The merchant and bank programs are continually listening to UDP sockets for SNPP messages. When a message is received, the appropriate library routine is called, which may in turn generate another message.

Two initial applications which will use SNPP are currently in the design stage. They will serve as a proof of concept for the protocol implementation, and will provide two different sets of requirements for the application interface.

The first application will address the problem of collecting small amounts of money for various social activities in our research group. Although this application is almost trivial, we believe it will not only provide an initial testing ground for most of SNPP, but will also make a small but notable improvement in a typical office environment.

The second application that we plan to design is the distribution of stock quotes and other financial information over the network. We plan to obtain the information from a commercial source, which charges a periodic fee for bulk access, and resell the data using SNPP on an individual quote basis. This will provide a much wider testing ground for the protocol, since both multiple banks and the holding of funds will be in use. We believe that there is a great amount of demand for this type of information on the Internet, and SNPP will for the first time provide a legal way to obtain it without having to dial up a commercial source directly.

10 Conclusions

We have described a simple, practical protocol for open network payment transactions. SNPP allows funds to be held during a sensitive period of a transaction between distrustful parties. At the same time it provides partitionability, so that unwanted real-time verification can be avoided, with the risks borne by the party that receives the benefits of improved efficiency.

We have employed the BAN Logic of Authentication to verify the correctness of SNPP. In our experience, the most useful part of the verification process was not the resulting proof, but the process of generating the idealized protocol. The act of abstracting away all implementation dependent details helped us uncover flaws in earlier versions of SNPP.

We expect that the availability of the protocol presented in this work could provide the material motivation for the development of many production quality services, and SNPP will thus serve to make the Internet a more useful, practical environment.

Appendix I

Complete *BAN* Proof

This proof uses the notation, postulates, and protocol steps listed in Section 8, pages 20-23.

When B_C receives message 3, the following statement holds, according to the definition of "**sees**":

$$B_C \textbf{ sees } \{N_C, \textbf{ HOLDMSG}\}_{K_C}$$

Since we have the assumption that B_C **believes** $C \overset{K_C}{\leftrightarrow} B_C$, the *message-meaning* rule applies, yielding

$$B_C \textbf{ believes } (C \textbf{ said } (N_C, \textbf{ HOLDMSG}))$$

Since we have assumed that B_C **believes fresh**(N_C), the *nonce-verification* rule applies, and yields

$$B_C \textbf{ believes } (C \textbf{ believes } (N_C, \textbf{ HOLDMSG}))$$

Breaking a conjunction,

$$B_C \textbf{ believes } (C \textbf{ believes HOLDMSG})$$

Since we have assumed that B_C **believes** that C **controls** hold-requests, and since **HOLDMSG** is an instance thereof,

$$B_C \textbf{ believes } (C \textbf{ controls HOLDMSG})$$

The *jurisdiction* rule now applies, yielding

$$B_C \textbf{ believes HOLDMSG}$$

Since B_C now believes the hold request, it will execute it, provided conditions such as fund availability are met.

After message 4 is received by B_M, the following statement holds, according to the definition of "**sees**":

$$B_M \textbf{ sees } \{N_{B_C}, \textbf{HELDMSG1}\}_{K_{B_C,B_M}}$$

Since we have the assumption that B_M **believes** $B_C \overset{K_{B_C,B_M}}{\longleftrightarrow} B_M$, the *message-meaning* rule applies, yielding

$$B_M \textbf{ believes } (B_C \textbf{ said } (N_{B_C}, \textbf{ HELDMSG1}))$$

Since we have assumed that B_M **believes fresh**(N_{B_C}), the *nonce-verification* rule applies, and yields

$$B_M \textbf{ believes } (B_C \textbf{ believes } (N_{B_C}, \textbf{ HELDMSG1}))$$

Breaking a conjunction,

$$B_M \textbf{ believes } (B_C \textbf{ believes HELDMSG1})$$

Since we have assumed that B_M **believes** that B_C **controls** held-messages, and since **HELDMSG1** is an instance thereof,

$$B_M \textbf{ believes } (B_C \textbf{ controls HELDMSG1})$$

The *jurisdiction* rule now applies, yielding

$$B_M \textbf{ believes HELDMSG1}$$

Since B_M now believes that B_C is holding the money, it can put its own guarantee behind the hold by sending message 5 back to the merchant.

After message 5 is received by the merchant, the following statement holds, according to the definition of "**sees**":

$$M \textbf{ sees } \{N_{B_M}, \textbf{HELDMSG2}\}_{K_{B_M}}$$

Since we have the assumption that M **believes** $B_M \stackrel{K_M}{\longleftrightarrow} M$, the *message-meaning* rule applies, yielding

$$M \textbf{ believes } (B_M \textbf{ said } (N_{B_M}, \textbf{ HELDMSG2}))$$

Since we have assumed that M **believes** **fresh**(N_{B_M}), the *nonce-verification* rule applies, and yields

$$M \textbf{ believes } (B_M \textbf{ believes } (N_{B_M}, \textbf{ HELDMSG2}))$$

Breaking a conjunction,

$$M \textbf{ believes } (B_M \textbf{ believes HELDMSG2})$$

Since we have assumed that M **believes** that B_M **controls** held-messages, and since **HELDMSG2** is an instance thereof,

$$M \textbf{ believes } (B_M \textbf{ controls HELDMSG2})$$

The *jurisdiction* rule now applies, yielding

$$M \textbf{ believes HELDMSG2}$$

The merchant now believes that the money is being held, and can safely send out the product.

The following are the idealized protocol steps for the **PAY** half of the protocol, which were not included in the body of the thesis:

1. $C \implies M \ : \ \{N_C, \textbf{PAYMSG}\}_{K_C}$

2. $M \Longrightarrow B_M$: $\{N_C, \textbf{PAYMSG}\}_{K_C}$

3. $B_M \Longrightarrow B_C$: $\{N_C, \textbf{PAYMSG}\}_{K_C}$

4. $B_C \Longrightarrow B_M$: $\{N_{B_C}, \textbf{PAIDMSG1}\}_{K_{B_C,B_M}}$

5. $B_M \Longrightarrow M$: $\{N_{B_M}, \textbf{PAIDMSG2}\}_{K_M}$

The following three assumptions were also omitted from the body of the thesis, since they deal only with the **PAY** half of the protocol:

1. B_C **believes** (C **controls** pay-requests)

2. B_M **believes** (B_C **controls** paid-messages)

3. M **believes** (B_M **controls** paid-messages)

When B_C receives message 3, the following statement holds, according to the definition of "**sees**":

$$B_C \textbf{ sees } \{N_C,\ \textbf{PAYMSG}\}_{K_C}$$

Since we have the assumption that B_C **believes** $C \overset{K_C}{\leftrightarrow} B_C$, the *message-meaning* rule applies, yielding

$$B_C \textbf{ believes } (C \textbf{ said } (N_C,\ \textbf{PAYMSG}))$$

Since we have assumed that B_C **believes** **fresh**(N_C), the *nonce-verification* rule applies, and yields

$$B_C \textbf{ believes } (C \textbf{ believes } (N_C, \textbf{ PAYMSG}))$$

Breaking a conjunction,

$$B_C \textbf{ believes } (C \textbf{ believes PAYMSG})$$

Since we have assumed that B_C **believes** that C **controls** pay-requests, and since **PAYMSG** is an instance thereof,

$$B_C \textbf{ believes } (C \textbf{ controls PAYMSG})$$

The *jurisdiction* rule now applies, yielding

$$B_C \textbf{ believes PAYMSG}$$

Since B_C now believes the pay request, it will execute it, provided conditions such as funds availability are met.

After message 4 is received by B_M, the following statement holds, according to the definition of "**sees**":

$$B_M \textbf{ sees } \{N_{B_C}, \textbf{PAIDMSG1}\}_{K_{B_C,B_M}}$$

Since we have the assumption that B_M **believes** $B_C \overset{K_{B_C,B_M}}{\longleftrightarrow} B_M$, the *message-meaning* rule applies, yielding

$$B_M \textbf{ believes } (B_C \textbf{ said } (N_{B_C}, \textbf{ PAIDMSG1}))$$

Since we have assumed that B_M **believes fresh**(N_{B_C}), the *nonce-verification* rule applies, and yields

$$B_M \textbf{ believes } (B_C \textbf{ believes } (N_{B_C}, \textbf{ PAIDMSG1}))$$

Breaking a conjunction,

$$B_M \textbf{ believes } (B_C \textbf{ believes PAIDMSG1})$$

Since we have assumed that B_M **believes** that B_C **controls** paid-messages, and since **PAIDMSG1** is an instance thereof,

$$B_M \textbf{ believes } (B_C \textbf{ controls PAIDMSG1})$$

The *jurisdiction* rule now applies, yielding

$$B_M \textbf{ believes PAIDMSG1}$$

Since B_M now believes that B_C will transfer the money, it can put its own guarantee behind the pay by sending message 5 back to the merchant.

After message 5 is received by the merchant, the following statement holds, according to the definition of "**sees**":

$$M \textbf{ sees } \{N_{B_M}, \textbf{PAIDMSG2}\}_{K_{B_M}}$$

Since we have the assumption that M **believes** $B_M \stackrel{K_M}{\longleftrightarrow} M$, the *message-meaning* rule applies, yielding

$$M \textbf{ believes } (B_M \textbf{ said } (N_{B_M}, \textbf{ PAIDMSG2}))$$

Since we have assumed that M **believes fresh**(N_{B_M}), the *nonce-verification* rule applies, and yields

$$M \textbf{ believes } (B_M \textbf{ believes } (N_{B_M}, \textbf{ PAIDMSG2}))$$

Breaking a conjunction,

$$M \textbf{ believes } (B_M \textbf{ believes PAIDMSG2})$$

Since we have assumed that M **believes** that B_M **controls** paid-messages, and since **PAIDMSG2** is an instance thereof,

$$M \textbf{ believes } (B_M \textbf{ controls PAIDMSG2})$$

The *jurisdiction* rule now applies, yielding

$$M \textbf{ believes PAIDMSG2}$$

The merchant now knows that the money has been transferred to her account, and can safely draw upon it.

References

[1] **M. Abadi, M. Burrows, C. Kaufman, B. Lampson,** "Authentication and Delegation with Smart-cards," *DEC Systems Research Center,* October 1990.

[2] **J. N. E. Bos, D. Chaum,** "SmartCash: a Practical Electronic Payment System," *Centrum Voor Wiskunde en Informatica,* Report CS-R9035, August 1990.

[3] **M. Burrows, M. Abadi, R. Needham,** "A Logic of Authentication," *Proceedings of the Royal Society of London,* Vol. 426, 1989.

[4] **L. Harn, T. Kiesler,** "Authenticated Group Key Distribution Scheme for a Large Distributed Network," *Proceedings 1989 IEEE Symposium on Security and Privacy,* May 1989.

[5] **R. Hirschfeld,** "Electronic Cash" Unpublished Manuscript, 1991.

[6] **P. Kirkman,** "Electronic Funds Transfer Systems," Basil Blackwell Ltd., Oxford, UK, 1987.

[7] **C. J. Mitchell, F. C. Piper,** "Key Storage in Secure Networks," *Discrete Applied Mathematics,* Vol. 21, No. 3, October 1988.

[8] **R. M. Needham, M. D. Schroeder,** "Using Encryption for Authentication in Large Networks of Computers," *Communications of the ACM,* Vol. 21, No. 12, December 1978.

[9] **D. M. Nessett,** "A Critique of the Burrows, Abadi, and Needham Logic," *ACM SIGOPS OS Review,* Vol. 24, No. 2, April 1990.

[10] **B. C. Neuman,** "Proxy-Based Authorization and Accounting for Distributed Systems," *University of Washington at Seattle,* Technical Report 91-02-01, March 1991.

[11] **E. Okamoto, K. Tanaka,** "Key Distribution System Based on Identification Information," *IEEE Journal on Selected Areas in Communications,* Vol. 7, No. 4, May 1989.

[12] **J. G. Steiner, C. Neuman, J. I. Schiller,** "Kerberos: an Authentication Service for Open Network Systems," *Proceedings of the Winter 1988 Usenix Conference,* February 1988.

[13] **V. L. Voydock, S. T. Kent,** "Security Mechanisms in High-level Network Protocols," *Computing Surveys,* Vol. 15, No. 2, June 1983.