A Peer-to-Peer Software Metering System

John Kelsey Bruce Schneier
{kelsey,schneier}@counterpane.com
Counterpane Systems, 101 East Minnehaha Parkway, Minneapolis, MN 55419

Abstract

We present two software-network payment systems, designed so that every user is capable of both buying and selling. One system uses online clearing; the other uses offline clearing.

1 Introduction

Internet commerce requires new payment methods, and several have already been developed [OO90, OO92, FY92, Bra93a, Bra93b, MN93, CR93, Fer94, LMP94, Oka95, MN95, BGH+95, ST95, TMSW95]. This paper describes a specific kind of payment system in which each user can transfer money to each other user: a peer-to-peer payment system. This can be thought of as a credit card account or a checking account. At regular intervals, these electronic payments are translated into real-world payments. Although this step is important to the users of these protocols, it's not a cryptographic operation at all; just a matter of executing EFTs or writing and mailing out checks.

This paper is organized as follows. Section 2 is a discussion of the objectives and design principles for this kind of system. In Section 3 we describe two example systems, one using online clearing, the other, offline clearing with public key signatures. Neither of these is meant as a final system, but both serve as good outlines to build a working system around.

2 Objectives, Design Principles, and Options

The ultimate objective is to have a payment system as handy for doing internet trade as cash, checks, or credit cards are for doing day-to-day trade. Whatever this is, it must meet the following criteria:

1. *Secure.* The system must be secure in the sense that it must not be possible to convince someone they have received money when they have not. It must also not be possible to forge a payment from someone else, or to replay payments. Finally, it mustn't be possible to violate other design specifications of the system, for example by tracing other peoples' payments.

2. *Cheap.* The system shouldn't require the user to purchase expensive equipment (other than a PC). The operating costs of the system should be low enough that nobody has to pay a large fixed monthly fee. Indeed, a small fixed monthly fee may be all anyone pays, if operation is cheap enough. In a computational sense, nobody should have to do too many complicated computations to pay someone a dime.

3. *Widely Available.* For maximum availability, the client (peer) software should be partially or completely available in source-code– participation in the payment system might be the paid for monthly, along with settling of bills as needed.

4. *Peer-to-Peer.* Each user should be able to either send or receive payments without complicated registration procedures.

There are several tradeoffs. For example, making a payment protocol computationally light usually means using few or no public key operations. However, this seems to require some kind of interaction with a central server, which drives up communications requirements and operating costs.

In this paper, we attempt to stay within the constraints of what is implementable in software on standard computers of today, using a good cryptographic function library. Individual users (peers) may have secrets, but these secrets have relevance only to those users. Only the bank is allowed to hold any global secrets.

2.1 Notation

Notation in the remainder of this document is as follows:

- $Enc_{K_A}(X)$ means that X is encrypted under key K_A, using some symmetric encryption algorithm.
- $Auth_{K_A}(X)$ means that X is authenticated under key K_A, using some symmetric message authentication algorithm.
- $EncAuth_{K_A}(X)$ means that X is authenticated and encrypted, using symmetric algorithms, under keys derived from K_A.
- $PKEnc_{K_A}(X)$ means that X is encrypted under public key K_A.
- $Sign_{K_A}(X)$ means that X is digitally signed under private key K_A.
- X, Y means that X is concatenated with Y.

3 Online Clearing Systems

Online clearing systems are simpler than offline systems, at least in principle, since both users can authenticate themselves to the Bank's trusted server instead of to each other. There are no certificates to worry about, and the most important security operations take place on a secured machine. Also, this system obviates the need for computationally expensive public-key operations (and the associated licensing issues). The cost of this is that the Bank has to maintain a constant online presence, and can easily become the bottleneck of the system.

In this section, we briefly discuss one example of an online system. In this system, we require the person accepting the payment to have a reasonably accurate clock, and to keep some short-term memory about transfers that have been accepted recently, to prevent trivial replay attacks. When Alice wants to transfer money to Carol, she first gets an electronic note of approval from the Bank for this transfer, which includes an expiration time (after which Carol should not accept it). Then, she sends this note to Carol to convince her that the transfer has indeed taken place. This turns out to be very close to the Kerberos protocol [Sch96].

3.1 Variables

1. S_A and S_C are Alice and Carol's sequence numbers. These must never repeat or go backwards—instead, they increment one value at a time. Suspicious jumping around by the sequence number must always be noted by the bank, and should initiate corrective action of some kind.
2. ID_A and ID_C are Alice and Carol's unique 64-bit ID numbers.
3. K_A and K_C are the keys shared between Alice and the Bank, and Carol and the Bank, respectively. The keys are derived based on a master key held by the bank, K_*, according to the relation
$$K_i = E_{K_*}(ID_i)$$
4. The audit-log is the log of all previous payments, kept by Alice's software. By including this hash, Alice commits to the current entries in the log; if she alters these later, it will be detected by the Bank [SK96].
5. A Timestamp is a 32-bit representation of a given time and date.

3.2 Basic Protocol: Alice Transfers Money to Carol

1. Alice forms

   U_0 = "Request for Transfer Authorization

   S_A = Current sequence number for Alice–incremented immediately

   $V = hash$(Audit Log)

   W = Amount of Transfer

   $X_0 = U_0, S_A, V, W, ID_C$

   K_0 = a random message key

   and sends to the Bank

   $M_0 = ID_A, \quad EncAuth_{K_A}(K_0), \quad EncAuth_{K_0}(X_0)$
2. The Bank receives and decrypts this. If the message doesn't decrypt or authenticate properly, the Bank responds with a simple error message. If the sequence number or hash of the audit log is wrong, then it must begin corrective action. This will be discussed further below. If the amount of the transfer is more than the

amount in Alice's account, the ID for Carol is invalid, or anything else is wrong with the message, then a simple error message is sent to Alice. However, if nothing is wrong with M_0, then the Bank forms

U_{1a} = "Transfer Authorization"
T_E = Expiration time of authorization
$Y_1 = U_{1a}, hash(M_0)ID_A, ID_C, W, T_E$
K_1 = a random message key
$Z_1 = EncAuth_{K_C}(K_1), EncAuth_{K_1}(Y_1)$
U_{1b} = "Transfer Verification"
S_C = Carol's current sequence number."
ID_A, ID_C, W from above step.
$X_1 = U_{1b}$, $hash(M_0)$, ID_A, ID_C, W, S_C, T_E
K_2 = a random message key

and sends back to Alice

$M_1 = EncAuth_{K_A}(K_2), EncAuth_{K_2}(X_1, Z_1)$

3. Alice receives this, and verifies everything she can verify. If there are no problems, she forms

U_2 = "Payment"
Z_1 from above.

and sends to Carol

$M_2 = U_2, Z_1$

At this point, Alice updates her audit log to include this transaction. This is done by appending ID_C, W to the previous contents of the audit log. In the event of any trouble verifying that this payment arrived, Alice and/or Carol contact the Bank.

4. Carol appends ID_A, W to her audit log, and adjusts her internally-carried balance by the amount of the transfer.

3.2.1 Security of the Protocol

1. *Replays*
   (a) The Bank would recognize any replay attempts immediately because of Alice's sequence number. Only an attacker that could alter or forge messages from Alice could replay a message with a new sequence number, without being caught at it.
   (b) Alice would recognize any replay attempts because the hash of her previous message is embedded inside the second protocol message.
   (c) Carol would recognize replay attempts directed at her by the replayed timestamp and incorrect sequence number. Note that there needs to be some room for error in both these values for Carol, because messages may not always arrive in the order they were sent, and because most computer clocks aren't highly accurate.
   (d) It's not possible to alter individual parts of transmitted messages because of the authentication being used. It's not possible to alter individual messages in a protocol because they typically contain the hash of the prior message.
2. *Forgeries* Forgeries should be very difficult to carry out, if the symmetric authentication scheme is acceptably strong.
3. *Information Leaks* Because of the use of a new message key for each message, it should be very hard for any information to leak. Within the message formats, only the ID of the person transferring money to someone else is leaked in any message.
4. *Other Attacks* There is a trivial attack in which Carol simply discards a payment to her by Alice, which means that Alice and the Bank will eventually need to resolve this. This is fundamentally a nuisance attack.

3.3 Secondary Protocol: Alice Reconciles

Occasionally, communications failures, implementation errors, or fraudulent transfers may leave Alice and the Bank unsynchronized. The solution is to go through a more complicated protocol to verify that all is well. This is also done from time to time to verify that Alice has received all the deposits that she should have received by now. During the ordinary transfer protocol, the Bank can require a reconciliation protocol before it will allow Alice to carry out any transfers, or Carol to receive any. This protocol is always required before depositing or withdrawing any money from the bank.

1. Alice forms

U_0 = "Request for Reconciliation"

R_0 = a random challenge

S_A = Alice's current internal sequence number

$X_0 = U_0, S_A$

K_0 = A random message key

she then sends to the Bank

$M_0 = ID_A, EncAuth_{K_A}(K_0), EncAuth_{K_0}(X_0)$.

2. The Bank verifies the authentication, and then forms

   U_1 = "Reconciliation Challenge"

   S_A^* = Bank's current idea about what S_A should be.

   K_1 = A random message key

   $X_1 = U_1, hash(M_0), S_A^*$

   it then sends to Alice

   $M_1 = EncAuth_{K_A}(K_1), EncAuth_{K_1}(X_1)$.

3. Alice verifies the authentication, and that the hash of the previous message is included. She then forms

   U_2 = "Reconciliation Response"

   $X_2 = U_2, hash(M_1)$,Full Authentication Log

   K_2 = a random message key

   $S_A = max(S_A, S_A^*) + 1$

   she then sends to the Bank

   $M_2 = EncAuth_{K_A}(K_2), EncAuth_{K_2}(X_2)$.

4. The Bank, after verifying all this information, either is or is not willing to let Alice start transferring money again. Thus, it forms

   U_{3a} = "Reconciliation Success Message"

   U_{3b} = "Reconcilliation Failure Message"

   K_3 = a random message key

   $X_{3a} = U_{3a}$, $hash(M_2)$, S_A, Account Balance

   $X_{3b} = U{+}3b, hash(M_2)$, Problem Description

   and sends to Alice, depending on the circumstances, one of the following:

   $M_{3a} = EncAuth_{K_A}(K_3), EncAuth_{K_3}(X_{3a})$.

   $M_{3b} = EncAuth_{K_A}(K_3), EncAuth_{K_3}(X_{3b})$.

At the end of this protocol, Alice and the Bank should have some common new sequence number, and either they should agree on what has happened, or there will be some kind of investigation going on, and Alice should know the basic kind of problem and what to do next.

3.4 What Can Go Wrong?

The most dire failure would involve a security breach at the Bank. Maintaining an authenticated, physically secure and separate log would be one way to ensure that a bank could recover from this kind of failure. Note that each payment from a user contains a current hash of her log.

The user's PC can lose security, in almost the same sense that someone can lose a credit card or checkbook. This should be rare, but it will generally have to be dealt with by humans. The software maintains a log, authenticated by chained hashes. Each payment commits to the current value of the log; that is, the hash of the log up until now.

The authentication and/or encryption functions can be broken. If this happens, the system must be recalled and fixed, and lots of money will be lost. This gives us a lot of incentive to choose our authentication and encryption functions well. For example, Triple-DES [NBS77] or Blowfish [Sch94] in CBC-mode might be used to encrypt, and a keyed hash might be made from SHA1 [NIST93], using a well-thought-out construction such as [PvO95].

4 An Off-line Payment System

The offline system can be imagined as a checking account: users are allowed to write checks for whatever's in their accounts. The software will not allow them to overdraw, but it is assumed that fraudulent users can change their software to allow this. We assume that we know how to deal with people who don't pay their bills: they can overdraw for a limited span of time, and the bank's collections department will wind up trying to get the money back from them. The offline system uses some public key operations instead of interactions with the

Bank. Although certificates are used, there is no CRL. Instead, certificates have a very short lifetime, perhaps a week. After that time, the user must connect to the bank to get a new certificate.

There are two routine operations: Payment and Reconciliation. In payment, one user (Alice) will pay $\$X$ to another user (Carol). In reconciliation, a user (Alice) will interact with the Bank to allow her to continue using the system. The system assumes that both Alice and Carol have reasonably accurate clocks.

4.1 Payment

In this protocol, Alice makes a payment to Carol by sending Carol a "draft." That is, a timestamped, digitally signed authorization for the bank to move $\$X$ from Alice's account into Carol's. Each draft has a unique identifying number, which prevents the bank from ever accepting replays, and gives Alice a way to refer to disputed payments.

Variables:

1. Timestamp is the current timestamp, in some standard format. The timestamp must never repeat, so it should include everything from centuries to seconds. An example might be "19951225010000".
2. The audit-log is the log of all previous payments. By including this, Alice commits to the current contents of her log each time she makes a payment. Later changes are detected by the Bank during reconciliation.
3. The Draft is the order specifying a draft sequence number, the account number of the payor, and the account number of the payee.
4. $Cert_A$ is Alice's certificate, attesting to the current valid linkage between her public signing key and her ability to write drafts on a given account id.
5. PK_A is Alice's public signing key.

This is the protocol by which Alice pays Carol:

1. Alice forms

   U_0 = "Check Transmission"
   T_0 = timestamp
   $V_0 = hash(audit - log)$
   $X_0 = U_0, V_0, T_0$, Draft
   $S_0 = Sign_{SK_A}(X_0)$
   K_0 = a random message key

   and sends to Carol

   $M_1 = Cert_A, PKE_{PK_C}(K_0), Enc_{K_0}(X_0, S_0)$

2. Carol verifies Alice's certificate and timestamp. She then verifies the signature on Alice's draft. If all is well, she knows that the payment has occurred.

This protocol is somewhat computationally expensive. Alice must perform a signature, and Carol must perform two verifications (though she may want to maintain lists of valid certificates, so she doesn't perform the certificate verification more often than necessary). Note that all payments from Alice to Carol appear in the clear; if a secure connection has already been made, then this will work well. If not, it may be necessary in some cases to establish a secure connection; this implies more public key operations for a strong key exchange. Also note that nothing keeps Carol from refusing to acknowledge Alice's payment; Alice can contact the bank and send a signed request to have a stop-payment put on that draft.

4.2 Deposit

Deposit simply consists of Carol sending to the Bank (perhaps by e-mail) the digitally signed draft from Alice. Since the draft names Carol, there is no risk of redirection. An active attack can prevent the bank from receiving the deposit. The Bank returns a receipt, also digitally signed. Because each draft must be different, there is no risk of replay attack, if things are done properly.

This is the protocol by which Carol deposits a draft:

1. Carol forms

   U_1 = "Deposit Request"
   $T = Timestamp$
   $V_1 = X_0, S_0$ from the previous protocol
   $X_1 = U_1, T, V_1$.
   $S_1 = Sign_{SK_C}(X_1)$
   K_1 = a random message key

and sends to the Bank

$$M_1 = PKE_{PK_C}(K_1), Enc_{K_1}(X_1, S_1)$$

2. The Bank verifies that all is well. Tf so, it forms

   U_2 = "Deposit Receipt"
   $R = Receipt$
   $X_2 = U_2, R, hash(M_1)$
   $S_2 = Sign_{SK_B}(X_2)$
   K_2 = a random message key

   and sends to Carol

$$M_2 = PKE_{PK_C}(K_2), Enc_{K_2}(X_2, S_2)$$

If Carol never receives M_1, she restarts the protocol later, or eventually notes it when she reconciles.

4.3 Reconciliation

Reconciliation occurs when Alice connects with the Bank to send in her logs, including copies of all drafts she has written, and to receive a new certificate. User certificates expire often.

1. The Bank forms

   U_3 = "Reconcilliation Request"
   R_3 = a random 64-bit value
   $X_3 = U_3, R_3$
   $S_3 = Sign_{SK_B}(X_3)$
   K_3 = a random message key

   and sends to Alice

$$M_3 = PKE_{PK_A}(K_3), Enc_{K_3}(X_3, S_3)$$

2. Alice forms

   R_4 = a random 64-bit value
   U_4 = "Reconcilliation Response"
   L = the log of payments and deposits since last reconciliation, noting any for which she received no receipt, and any for which she wishes to dispute payment.
   $X_4 = U_4, hash(M_3), R_4, L$
   $S_4 = Sign_{SK_A}(X_4)$
   K_4 = a random message key

   and sends to the Bank

$$M_4 = PKE_{PK_B}(K_4), Enc_{K_4}(X_4, S_4).$$

3. The Bank verifies that all is well. If so, it responds by forming

   U_5 = "Reconciliation Receipt"
   C_A = New Certificate for Alice with a new timestamp
   $X_5 = U_5, hash(M_4), C_A$, Ending Balance, Timestamp
   $S_5 = Sign_{SK_B}(X_5)$
   K_5 = a random message key

   and sends to Alice

$$M_5 = PKE_{PK_A}(K_5), E_{K_5}(X_5, S_5)$$

If there are disputed payments or other problems, there will be a request for more information sent. This should lead to an online session, telephone call, or face-to-face discussion of the user's complaint. Dispute resolution falls outside of the scope of the payment system.

4.4 Error Recovery, Fraud Detection, and Auditing

A continual audit trail is kept in the user's PC, and its current value is committed to each time a payment is made. Forged transactions will be caught during reconciliation, as will almost all other problems. People who overdraw their accounts, exceed their credit lines, or fail to pay their bills have either had a software flaw, or hacked their software to do these things. They will not be issued a new certificate until they resolve the problem. This limits damage from individual security failures to a small period of time, perhaps no more than a week.

4.5 What Can Go Wrong?

Compromise of the private signing key used to create certificates is probably the most catastrophic thing that could happen. This must be guarded against strongly, perhaps maintaining the key in a tamper-resistant token under good physical security. Compromise of this key would allow an attacker to write an endless stream of bad checks on anyone's account.

The user's PC can lose security, in almost the same sense that someone can lose a credit card or checkbook. This should be rare, but it will generally have

to be dealt with by humans. The software maintains a log, authenticated by chained hashes. Each payment commits to the current value of the log: i.e., the hash of the log up until now. Since certificates expire often, this will be dealt with soon after it's discovered by freezing the account that has been compromised until the problem can be dealt with. In this system, another thing that can happen is that a user's clock can be fouled up, to trick him into accepting a stale certificate. Such attacks must be guarded against.

The authentication, signing, and/or encryption functions can be broken. If this happens, the system must be recalled and fixed, and lots of money will be lost. This gives us a lot of incentive to choose our functions well. For example, we might use 1280-bit RSA [RSA78] SHA1 [NIST93] for our signatures.

4.6 Additional Comments and Extensions

Note that it is possible to give the payor anonymity from the payee. This is done by giving each user a large number of IDs, each with a different public signing key and certificate. This gives no anonymity from the bank.

5 Retrofitting Existing Systems

There are several reasonably good methods of paying for things available now. Small variations in these could be adapted to this kind of an applications. The advantages of using an already fielded system are that there is no need to build the system's infrastructure, and that a fielded system may already have had many of its flaws worked out. Among the possible disadvantages are that many currently-fielded payments systems don't meet the specific needs of this application, and that there are often hardware requirements and licensing fees.

- Digicash is payer-anonymous electronic cash. It is currently available. The cost per transaction appears to be too high, and there are some practical security problems with any payer-anonymous system that need to be worked out, but this kind of system might be worth using for this application at some point.
- Some schemes simply transfer users' credit card numbers around. These have obvious security problems involving replay and forgery attacks. However, one way to implement any of the systems discussed above would be for the bank to bill users' credit card for any money owed. To avoid high transaction costs, the bank would bundle many small transactions per month before billing them—perhaps only in $50 increments—from the user's card. The advantage of this would be in low start-up costs. Ideally, the system would also be able to credit money to their credit cards, or transfer it into their bank accounts. This would simplify the mechanics of running this kind of system.
- There are micropayment schemes, such as Millicent, Payword, and Micromint, that allow for small monetary values to be transferred among parties. However, micropayment schemes generally have to solve somewhat different problems than a peer-to-peer metering system, so they make somewhat different tradeoffs.

6 Conclusion

The future of the Internet is one where many people are both producers and consumers of information. Any payment system needs to reflect this. This work, while no means complete, shows the kind of directions necessary for peer-to-peer payment systems.

References

[BGH+95] M. Bellare, J. A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner, "iKP - A Family of Secure Electronic Payment Protocols", *The First USENIX Workshop on Electronic Commerce,* USENIX Association, 1995, pp. 89–106.

[Bra93a] S. Brands, "Untraceable Off-line Cash in Wallets with Observers," *Advances in Cryptology—CRYPTO '93 Proceedings,* Springer-Verlag, 1994 pp. 302–318.

[Bra93b] S. Brands, "An Efficient Off-line Electronic Cash Systems Based on the Representation Problem," C.W.I. Technical Report CS-T9323, 1993.

[CR93] CitiBank and S. S. Rosen, "Electronic-Monetary System," International Publication Number WO 93/10503; May 27 1993.

[Fer94] N. Ferguson, "Extensions of Single-Term Coins," *Advances in Cryptology—CRYPTO '93 Proceedings*, Springer-Verlag, 1994, pp. 292–301.

[FY92] M. Franklin and M. Yung, "Towards Provably Secure Efficient Electronic Cash," Columbia Univ. Dept of C.S. TR CUCS-018-92, April 24, 1992. (Also in Icalp-93, July 93, Lund Sweden, LNCS Springer-Verlag).

[LMP94] S. H. Low, N. F. Maxemchuk and S. Paul, "Anonymous Credit Cards," *The Second ACM Conference on Computer and Communications Security,* ACM Press, 1994, pp. 108–117.

[MN93] G. Medvinsky and B. C. Neuman, "Netcash: A Design for Practical Electronic Currency on the Internet," *The First ACM Conference on Computer and Communications Security,* ACM Press, 1993, pp. 102–106.

[NBS77] National Bureau of Standards, NBS FIPS PUB 46, "Data Encryption Standard," National Bureau of Standards, U.S. Department of Commerce, Jan 1977.

[MN95] B. C. Neuman and G. Medvinsky, "Requirements for Network Payment: The NetCheque^{TM} Perspective," Compcon '95, pp. 32–36

[NIST93] National Institute of Standards and Technology, NIST FIPS PUB 180, "Secure Hash Standard," U.S. Department of Commerce, May 93.

[Oka95] T. Okamoto, "An Efficient Divisible Electronic Cash Scheme," *Advances in Cryptology—CRYPTO '95 Proceedings*, Springer-Verlag, 1995, pp. 438–451.

[OO90] T. Okamoto and K. Ohta, "Disposable Zero-Knowledge Authentication and Their Applications to Untraceable Electronic Cash," *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 481–496

[OO92] T. Okamoto and K. Ohta, "Universal Electronic Cash," *Advances in Cryptology—CRYPTO '91 Proceedings*, Springer-Verlag, 1992, pp. 324–337.

[PvO95] B. Preneel and P. C. van Oorschot, "MDX-MAC and Building Fast MACs from Hash Functions," *Advances in Cryptology—CRYPTO '95 Proceedings*, Springer-Verlag, 1995, pp. 1–14.

[RSA78] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, v. 21, n. 2, Feb 1978, pp. 120-126.

[Sch94] B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)," *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Springer-Verlag, 1994, pp. 191–204.

[Sch96] B. Schneier, *Applied Cryptography, Second Edition*, John Wiley & Sons, 1996.

[SK96] B. Schneier, J. Kelsey, "Automatic Event-Stream Notarization Using Digital Signatures," in *Advances in Cryptology, Proceedings of the Cambridge Protocols Workshop 96*, Springer-Verlag, 1996, pp. in preparation.

[ST95] M. Sirbu and J. D. Tygar, "NetBill: An Internet Commerce System Optimized for Network Delivered Services," Compcon '95, pp. 20–25.

[TMSW95] J. M. Tenenbaum, C. Medich, A. M. Schiffman, and W. T. Wong, "CommerceNet: Spontaneous Electronic Commerce on the Internet," Compcon '95, pp. 38–43