KEY MANAGEMENT FOR SECURE ELECTRONIC FUNDS TRANSFER IN A RETAIL ENVIRONMENT

Henry Beker and Michael Walker

Racal Research Limited,
Worton Drive,
Worton Grange Industrial Estate,
Reading, Berkshire, England.

1. INTRODUCTION

In this paper we consider a system whose function is to enable users to pay for goods or services by direct electronic transfer of funds. The system consists of terminals, located at retail outlets, which can communicate with acquirers representing various financial institutions.

Each user of the system has a plastic card and a personal identification number (PIN) issued by a financial institution represented in the system. Affixed to the plastic card is a magnetic stripe, which bears the card holder's personal account number as well as other data such as the expiry date of the card. The PIN is typically four decimal digits long, and is effectively the card holder's electronic signature. He is expected to treat it as such, and refrain from divulging it to unauthorised third parties. Of course, the card holder in turn expects that his PIN will be adequately protected by any authorised body which has knowledge of it.

Throughout this paper we make the simplifying assumption that an acquirer holds a complete data base for every card issued by the financial institutions it represents. In particular, an acquirer has a record of corresponding PINs and card data for all of those cards it is authorised to handle. In addition, we assume that the acquirer is in a position to endorse every transaction made with these cards.

When a card holder wishes to make payment for a purchase from the retailer, the plastic card is presented, and the data encoded on the magnetic stripe is read by the terminal. This gives the terminal data related to the card holder, and identifies the acquirer for the particular transaction. The card holder separately enters his PIN. The retailer enters the details of the purchase, and the terminal then communicates with the acquirer, whose function is to ratify the transaction. This entails checking that the card is valid, that the account contains sufficient funds for the purchase, and that the PIN and card do in fact correspond. It is in this sense that the PIN acts as an electronic signature to authenticate the card holder to the acquirer.

Having completed these tasks, the acquirer informs the terminal that the purchase can proceed (or otherwise), and then arranges for the appropriate funds to be credited to the retailer from the card holder's account.

Clearly, it is essential that the PIN should be kept secret, that the transaction messages should be protected against corruption or intentional change during transmission, and that all parties should authenticate each other. These security requirements can all be satisfied by using cryptographic functions based on block ciphers such as the Data Encryption Algorithm [3]. Indeed, techniques to achieve this are published in a number of papers, notably [4] and [5], which describe PIN encryption and message protection respectively. The main problem is not the performance of the security functions themselves, but rather management of the enciphering keys.

To understand some of the problems involved with key management, let us reconsider our entire system. Clearly, security will certainly be enhanced if all encryption and authentication takes place on an end-to-end basis without additional parties becoming involved. This also has the added advantage that it makes the system network independent, which means that in theory any transmission medium can be used, and in practise it is possible to use an alternative media should the primary one fail. Now our system may well have hundreds of thousands of terminals communicating with a hundred or so acquirers. Thus the problem of distributing the multitude of cryptographic keys is paramount. Consequently, if end-to-end security is to be used, it is obviously desirable that the system is equipped with a procedure for automatically updating the keys. Of course when attempting to design such a procedure, one must bear in mind that it should not be possible for one acquirer to inadvertently compromise the keys of another. Taking this consideration a little further, it is clearly desirable that disclosure of a key should compromise at most one transaction. A system with this feature offers scant reward for anyone who manages to gain knowledge of a particular key. Finally, it must be stressed that our terminals are at most tamper-resistant and certainly not tamper-proof. In particular, a key housed for any length of time in a terminal cannot really be considered safe enough to be used to encipher a PIN. Once again, this points to the desirability of a system which automatically generates a fresh key for each transaction.

The fresh key per transaction, or transaction key, approach was introduced in [1] as a way of overcoming some of these key management problems. The present article expands on the ideas of that paper, and develops a protocol for key management in the electronic transfer of funds system defined above. It should be stressed that the techniques are applicable to more general systems than the one chosen here. Our choice was made on the basis of requiring a system which reflected many of the central problems of key management, without being so complex that it overshadowed the salient features of the scheme. For a discussion in a wider context the reader is referred to [2]. For simplicity of description, the protocols are defined for a basic request-response message flow between a terminal and an acquirer. The scheme is such that confirmation of (non-) completion at a terminal of a particular transaction is automatically conveyed by the next request from that terminal to the acquirer. This does not mean that the system will not support a confirmation message within a transaction. Indeed, the protocols are readily adapted to accommodate a far more comprehensive dialogue than the one described here.

Throughout the paper, we shall base all cryptographic functions needed to describe our key management protocols on the Data Encryption Algorithm (DEA). Of course, this is just a convenience, and it is not necessary to appreciate how the algorithm works. All that is required is to know that it transforms 64 bit blocks of clear text to 64 bit blocks of cipher text under control of an enciphering key. The key is also 64 bits in length, but only 56 of the bits are actually used by the enciphering algorithm. We shall denote the clear text input to the algorithm by DATA, the resulting cipher text block by CIPHER and the controlling key by KEY. Thus, for our purposes, the enciphering algorithm E is described by

$$E(KEY) \; : \; DATA \longmapsto CIPHER$$

Having established this notation, we return to our system and describe how the transaction keys are generated.

2. TRANSACTION KEYS

We make use of the well known concept of including some card holder dependent data on the magnetic stripe of the card, and refer to this data as the <u>card key</u>. This data is read by the terminal but is not itself transmitted. In addition, the terminals contain a <u>key register</u> for each acquirer with whom it is authorised to communicate.

Let us assume that our system is up and running, and that a card holder, wishing to pay for a purchase, presents his plastic card to the terminal. The terminal then reads the card key and the card holder's personal account number (PAN), and identifies the acquirer for the particular card. The <u>transaction key</u> is then generated at the terminal as a one-way function of the card key and the value in the key register for the particular acquirer.

Assuming the one-way function is to be based upon the DEA, then the transaction key might be generated as follows:

$$\begin{array}{rcl} \text{DATA} & \longleftarrow & \text{card key} \\ \text{KEY} & \longleftarrow & \text{key register value} \\ \text{transaction key} & \longleftarrow & \text{CIPHER} \oplus \text{DATA} \end{array}$$

Of course, the terminal must provide the acquirer with sufficient information to generate the transaction key at his end. To accomplish this, the request message from the terminal includes in clear text the card holder's PAN and the terminal's identification. The acquirer maintains a key register for each terminal in the system, and the value in the register for a particular terminal agrees with the value held by that terminal in its key register for the acquirer. Since by hypothesis the acquirer holds a complete data base for every card it is authorised to handle, the acquirer has a list of corresponding card keys and PANs. Thus the acquirer is able to construct the transaction key by identifying the card key from the PAN and the key register value from the terminal identification.

We have defined the transaction key as a function of two independent variables, the card key and the key register value, which is generated by both parties precisely when it is needed. In general, it is not possible to predict its value in advance because of the element of randomness provided by the card key. Naturally, we must update the key register value at the end of the transaction, and we also wish to make this updating a random process. As we shall see, this second element of randomness is provided by the unpredictability of the messages exchanged during the transaction.

3. THE REQUEST MESSAGE

Once the terminal has read the card and constructed the transaction key, details of the transaction are entered and a request message is compiled. This message must include the card holder's PAN and the terminal identification, both in clear text. In our system it will also include the card holder's PIN (or PIN offset), which should be separately entered and enciphered under the transaction key before being inserted in the message. It may also include other cipher blocks as well.

Having compiled the request message, the terminal must now add an ingredient which protects it against change. In addition, the terminal must have a procedure to authenticate the acquirer, and a way of checking that the acquirer's response does indeed pertain to the particular request. These three authentication checks are achieved by generating a message authentication block (MAB) under control of the transaction key. Generation of this block follows the procedure described in [5] for constructing a message authentication code (MAC). The message is divided into n blocks

$$B_1, B_2, \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, B_n$$

each 64 bits in length. The DEA is then used to process these blocks sequentially under control of the transaction key. More precisely

$$\text{KEY} \leftarrow \text{transaction key}$$

and n data blocks

DATA1, DATA2, ..., DATAn

are sequentially processed under KEY to produce cipher blocks

CIPHER1, CIPHER2, ..., CIPHERn

where

$$\text{DATA1} = B_1$$

and

$$\text{DATAj} = \text{CIPHER}(j-1) \oplus Bj$$

for j = 2, 3, ..., n. The request MAB is defined to be the final cipher block CIPHERn.

The left hand half of the MAB forms the request MAC. This is inserted into the message before transmission, and allows the acquirer to verify that the message has not been changed during transmission. The remaining 32 bits of the MAB form the request residue. This is retained by the terminal, and used later to authenticate the acquirer as the originator of the response message, and to confirm that the response corresponds to the request. It is also used in the key register updating procedure.

When the acquirer receives the request message, it generates the transaction key, removes the request MAC from the message, and then uses the algorithm described above to generate a MAB for the truncated message. The left hand half of the MAB is then compared with the request MAC retained from the received message in order to confirm the message integrity. If the two do in fact agree, then this also authenticates the terminal to the acquirer because their key register values must be identical. Once the request message has been authenticated, the right hand half of the MAB is retained by the acquirer as the request residue.

4. THE RESPONSE MESSAGE

When the acquirer has finished checking the transaction details, he prepares a response message, and generates a MAB under control of the transaction key. The MAB generation procedure for the response message is slightly different from that described for the request message. The acquirer first adds the request residue to the beginning of the response, and then constructs a MAB for the extended message. The rationale for constructing the response MAB on the extended message is to tie the response to the request which prompted it. This enables the terminal to authenticate the acquirer as the originator of the response, as well as to confirm that the response does indeed correspond to the request.

The left hand half of the response MAB is the response MAC. This is inserted into the response message after the request residue has been removed, and is used by the terminal to authenticate the message. The other half of the response MAB is called the response residue. This is retained by the acquirer, along with the request residue, to be used to update the key register.

When the terminal receives the response message, it removes the response MAC, adds the request residue to the beginning of the message, and then generates a MAB for the extended message. The left hand half of the MAB is then compared with the request MAC retained from the incoming response message. If the two agree, then the terminal has authenticated both the message and the acquirer and confirmed that the received response corresponds to the original request message. Once authentication has been completed, the terminal retains the right hand half of the MAB as the response residue, and completes its end of the transaction.

5. KEY REGISTER UPDATING

After the acquirer has transmitted the response message, the transaction key is destroyed, the current key register value is stored, and the key register is updated. The reason for retaining a copy of the key register value will be explained later. First, we describe the key register updating procedure.

The new value of the key register is a one-way function of the current value, the request residue and the response residue. If we use the same one-way function as we used to generate the transaction key, then the updating procedure may be defined by:

$$\begin{aligned} \text{DATA} &\longleftarrow \text{(request residue, response residue)} \\ \text{KEY} &\longleftarrow \text{key register value} \\ \text{key register value} &\longleftarrow \text{CIPHER} \oplus \text{DATA}, \end{aligned}$$

where the input to DATA is a concatenation of the request and response residues. Thus the new key register value depends upon the old value and the message residues. Since the message residues depend upon the messages exchanged and the transaction key, which itself depends upon the card key, the new key register value is in general quite unpredictable.

The terminal destroys the transaction key and updates its own key register value after it has authenticated the response message; the updating procedure being identical to that of the acquirer. Of course, if the response message fails to reach the terminal or fails to be authenticated, then the terminal's key register value remains unaltered, although the acquirer's has already been updated. It is precisely for this reason that the acquirer retains a copy of the previous value in its key register, for this enables the system to recover from the situation.

Suppose then that the acquirer has updated its key register, but the terminal has failed to do so. On receipt of the next request from the terminal, the acquirer constructs a new transaction key, and then attempts to authenticate the message. Naturally, this will fail because the terminal is still using the old key register value. The acquirer then replaces its key register value by the old value it has retained, and generates a transaction key based on this value. If authentication is now successful, then the acquirer recognises that the previous transaction did not complete at the terminal, and can take the necessary action. Moreover, the terminal's and acquirer's key registers now agree, so that synchronisation is recovered and the current transaction can proceed as normal.

The above discussion highlights one other feature of the system. Completion of a transaction at a particular terminal is confirmed to the acquirer by successful authentication of the next request received from that terminal.

6. CONCLUSION

There are a number of points to note regarding the key management scheme described in this paper. First, the transaction key is end-to-end and unique to the particular transaction. Even if an unauthorised person gained knowledge of a transaction key and the card key of the next card presented at the terminal for use with the same acquirer, this would not be sufficient to deduce the next transaction key. Similarly, it is not possible to deduce anything about the previous transaction with that acquirer. Thus the rewards for breaking a single key are indeed small. Secondly, key management is automatic, and a transaction key is unpredictable (because it depends on a card key and the value in a key register, and this value depends upon the previous value, the previous card key and the messages exchanged during the previous transaction). Thirdly, confirmation that a transaction completed at a terminal is inherent in the next communication between the terminal and acquirer. Fourthly, it should be noted that a log-in is not required, and the system does not need to be shut down for the purpose of distributing new keys. Fifthly, even if someone breaks into a tamper-resistant terminal and obtains the key register values for some acquirers, the information is useless to them just as soon as bona-fide cards are presented at the terminal for use with these acquirers. This removes some of the need for high physical security of the terminals.

The reader will probably have noted that we have described protocols for a system which is already operational, but have not mentioned how the system is initialised. This could be handled in several ways, and we shall make three suggestions. First, it is not inconceivable that a public key cryptosystem could be used to provide initial values for all key registers. Alternatively, each acquirer could insist that when a terminal is installed a test transmission using a test-card should preceed all other transactions. This same test-card might also be used to re-initialise in the event of a catastrophic failure. A third possibility is that an acquirer might well simply choose to ignore the problem, bearing in mind that once a bona fide card is presented the terminal - acquirer link attains full security.

Finally, we mention once again that the key management scheme outlined in this paper is applicable to more general systems than the one we have described. The techniques can be adapted to cover the situation where the acquirer does not hold a complete data base for each card it is authorised to handle, and to provide for a more extensive dialogue between terminal and acquirer.

REFERENCES:

1. H.J. Beker, J.M.K. Friend, P.W. Halliden; Simplifying key management in electronic fund transfer point of sale systems, Electronics Letters Vol.19 No.12 (1983), 442-444.
2. H.J. Beker; Secure electronic funds transfer in a retail environment; Networks 84, Proceedings of the European Computer Communications Conference (1984), 263-270.
3. American National Standard; "Data Encryption Algorithm", ANSI X3.92 (1981).
4. American National Standard; "Pin Management and Security", ANSI X9.8 (1982).
5. American National Standard; "Financial Institution Message Authentication", ANSI X9.9 (1982).