Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Commerce Research and Applications 4 (2005) 413–426

Electronic Commerce Research and Applications

www.elsevier.com/locate/ecra

Securing credit card transactions with one-time payment scheme ☆

Yingjiu Li [a,*], Xinwen Zhang [b]

[a] School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902
[b] Lab for Information Security Technology, George Mason University, Fairfax, VA 22030, USA

Received 5 October 2004; received in revised form 7 January 2005; accepted 1 June 2005
Available online 21 July 2005

Abstract

Traditional credit card payment is not secure against credit card frauds because an attacker can easily know a semi-secret credit card number that is repetitively used. Recently one-time transaction number has been proposed by some researchers and credit card companies to enhance the security in credit card payment. Following this idea, we present a practical security enhancement scheme for one-time credit card payment. In our scheme, a hash function is used in generation of one-time credit card numbers with a secret only known to the card holder and issuer. Compared with related work, our scheme places less burden on credit card issuers, and can be easily deployed in on-line or off-line payment scenarios. Analysis and simulation show that the time and space complexity is affordable to the card issuer with desired security features.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Credit card transaction; Credit card fraud; Security

1. Introduction

Credit card frauds have caused millions of dollars loss each year and exposed the security weaknesses in traditional credit card processing system [2]. In such system, a customer (i.e., credit card holder) repetitively uses a fixed credit card number as well as personal identifying information in all transactions. Because this credit card number is "sticky", it is relatively easy for an attacker to steal

☆ A preliminary version of this paper appeared in the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications, pages 40–47, Boston, USA, March 2004.

* Corresponding author. Tel.: +65 6828 0913; fax: +65 6828 0919.

E-mail addresses: yjli@smu.edu.sg (Y. Li), xzhang6@gmu.edu (X. Zhang).

1567-4223/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.elerap.2005.06.002

it with intention to commit illegal activities. Some common ways to commit credit card fraud include:

- *Shoulder surfing*: An attacker watches a customer from a nearby location as the customer punches in his credit card number. If the customer is giving his credit card number over the phone (e.g., to a hotel or car rental company), the attacker may listen to the conversation so as to get credit card information.
- *Dumpster diving*: An attacker goes through a customer's garbage cans or trash bins to obtain copies of credit card statements.
- *Packet intercepting*: An attacker sniffs some e-commerce packets during on-line credit card payment. In some cases, the attacker does not need to break down the possibly encrypted packets (e.g., over Secure Socket Layer), but fools the customer into thinking that he or she is visiting an intended site but actually the attacker's spoofing one.
- *Database stealing*: To encourage purchasing, many merchants (who provide services to customers) choose to store their customers' credit card information in online databases. Recent news reported that attackers could break into merchants' web sites and steal millions of credit card numbers [1].

Not only does the credit card fraud cause millions of dollars loss each year, but also causes significant worry among customers. According to a recent study conducted by Opinion Research Corporation, it causes more worry than the war in Iraq in terms its impacts on customers' awareness of security issues [5].

1.1. Evaluation criteria

Many efforts have been made so far to thwart credit card fraud. Before we look into them, we summarize the evaluation criteria that has been proposed by Shamir [10], Rubin and Wright [7] for secure credit card payment.

- *Ease of deployment*: The system should be easy to deploy in real-world settings. There should be few additional requirements on current infrastructure and communication protocols. Card issuers should be able to handle most of the deployment tasks, without placing unreasonable burdens on merchants and customers. Even for the card issuers, the additional requirements on equipment should be tolerable compared with the benefits obtained from security enhancement.
- *Ease of use*: The importance of ease of use cannot be overstated since it is the customers who use the payment system. The customers should feel convenient in all payment scenarios.
- *Security*: A secure payment system should address real security concerns thus overcome customers' psychological fear due to credit card fraud. The security of the system may not be perfect, but it should be good enough to protect customers in all payment scenarios.

1.2. Related work

A previous effort to thwarting credit card fraud led to the development of Secure Electronic Transactions (SET) protocol [9]. SET was designed to protect credit card information from various attacks in on-line environment. Unfortunately, SET never succeeded in the marketplace because of its high overhead and additional requirement of public key infrastructure (PKI).

Among hundreds of other solutions [4] that have been proposed (most of them failed or remain untested), credit card payment over Secure Socket Layer (SSL) is the only one that is widely used in e-commerce nowadays. SSL [3] provides encryption channel to transmit credit card numbers; it also provides server authentication to identify merchants. While a flawless implementation of SSL thwarts packet intercepting, it has no effect on other credit card frauds such as shoulder surfing, dumpster diving and database stealing.

In terms of those evaluation criteria mentioned in Section 1.1, SET addresses the security concerns but fails to satisfy the requirements on ease of deployment and ease of use. On the other hand, SSL solution satisfies the requirements on ease of deployment and ease of use; however, it does not address the security concerns in all scenarios.

Recently, the concept of *one-time credit card transaction number* (also called disposable number, single-use number, nonce, or token) has been proposed. One-time credit card transaction number, CCT for short, is designed for single use. After its use, it does not matter whether it is learned by attackers.

Most of the existing solutions (e.g., American Express' Private Payments [6] and Shamir's SecureClick [10]) require CCTs be generated online; that is, during or shortly before a credit card transaction, a customer must have an on-line secure interaction with the card issuer so as to get a CCT. The card issuer associates the CCT with the customer in a stored database and therefore can verify it when an involving merchant sends the CCT for clearance.

As indicated by Rubin and Wright [7], these solutions augment traditional credit card transaction with an additional connection between customer and server (card issuer). The customer–server communication must be secured, typically by SSL; otherwise CCT can be learned by an attacker in the middle of or before its use. With a large number of customers connecting to the server with SSL simultaneously, the performance of the server will become a critical problem. In such case, the customers have to be patient while waiting for CCTs. In addition, such centralized solution with SSL does not scale well. A server with pure function of collecting credit card numbers is dangerous due to vulnerabilities such as single point failure, web site spoof, and DNS redirection.

We also note that these solutions are mainly designed for web payment. In other payment scenarios such as on-site shopping and telephone payment, it may not be always possible for a customer to interact with credit card issuer. To facilitate payment, many existing one-time solutions allow customers to use fixed credit card numbers as in traditional credit card payment. However, such "mixed" system is subject to existing attacks such as shoulder surfing, dumpster diving, and database stealing.

To bring a remedy to these problems, Rubin and Wright [7] proposed an off-line scheme for generating CCTs (called limited-use credit card numbers or tokens), without requiring the interaction between customer and card issuer. In their scheme, card issuer and customer share a long-term secret key. Before each transaction, the customer generates a CCT by encrypting a set of possible restrictions. The restrictions describe the purchase in terms of expense, time, merchant and etc. The customer then sends his CCT and identifying information to the card issuer (through a merchant) for verification. Upon receiving these, the card issuer locates the long term secret key according to the identifying information, decrypts the CCT, and verifies the purchase.

This solution does not require an interaction between customer and card issuer before each transaction; however, it requires well designed user interface on the customer side to help the selection of restrictions for encryption. It also burdens the server with decryption process and restriction management. In addition, it is not very clear how to apply standard encryption functions such as AES in this case if the CCTs are restricted to at most 16 digit credit card numbers [7].

In order to use this payment scheme, a customer must have access to a computer or specific device that can encrypt restrictions. A transaction time-stamp must be included in the encryption otherwise a generated CCT may not be unique. Certainly this requires application support. This scheme may be suitable for web payment; it may not be affordable in other payment scenarios.

1.3. Our solution

In our solution, we use hashing rather than encryption for off-line generation of CCTs. Each CCT is generated by hashing its previous CCT and a shared secret. The secret is a binary string known to the card issuer and is embedded into a customer's physical card.

A small chip is embedded into each credit card so as to perform the hash computation and store the previous CCT. To facilitate credit card transactions, a smart card reader is needed in most scenarios to empower the generation of a CCT. Note that the hash computation is simple; the techniques for producing chipped cards and smart card readers are mature and cheap. The smart card readers can be made publicly available (similar to

American Express' chipped Blue cards and free-distributed smart card readers).

On the customer side, what a customer needs to do is to insert his physical card into any smart card reader, and transmit a generated CCT to a merchant. The hash computation involved is simple and fast and no interaction between the customer and the card issuer is required. The whole process is as convenient as in traditional credit card payment provided that chipped cards and smart card readers are available.

Once a CCT (possibly with a customer's identifying information) reaches the card issuer for clearance, the CCT is verified if it matches a hash value computed from a previously verified CCT and the secret of that customer. Because of the existence of delayed verifications, a customer's CCTs may not arrive in the same order as they are generated; therefore, the card issuer needs to maintain a queue of CCTs for correct verification. Our study shows that the average length of the queue is small. Consequently, the time and space complexity for CCT verification increases little compared with traditional credit card payment.

Compared with Rubin and Wright's off-line scheme, our scheme places less burdens on the card issuer because the hash computation can be easily customized to generate fixed-length (e.g., 16 digits) CCTs. Also customers are not required to have application support for encrypting transaction time stamps and selecting among many restrictions.

In terms of security, our solution sticks to the concept of one-time payment idea. Unlike "mixed systems", our scheme is secure against most common credit card frauds such as shoulder surfing, dumpster diving, packet intercepting and database stealing.

1.4. Organization

The rest of the paper is organized as follows. Section 2 describes our customer payment scheme in various payment scenarios. Section 3 presents our verification scheme including verification algorithm, system simulation, and complexity analysis. In Section 4, we analyze the security aspects of our scheme. Section 5 discusses several implementation options and compares our scheme with PKI-based approach. Finally, Section 6 concludes the paper.

2. Customer payment scheme

In this section, we describe customer payment scheme in different payment scenarios. For convenience, Fig. 1 gives the notation that will be used in this paper.

2.1. Credit card

Traditional credit card consists of a fixed *credit card number* and other information such as name and expiration date. In our scheme, the credit card consists of two additional elements: (i) a secret (i.e., binary string) that never leaves the physical card and never changes; (ii) a one-time transaction number (CCT) that will be used in a single credit card transaction. The physical card is embedded with a small chip that stores both secret and CCT.

When a credit card is issued, an initial CCT is stored in the card (therefore, the card issuer knows the secret and the initial CCT for each customer). For each transaction to be processed, a new CCT is computed from the previous CCT and the secret using a cryptographic hash function $\mathscr{H}$:

$$T_{\mathrm{new}} = \mathscr{H}(T_{\mathrm{cur}} \| S), \tag{1}$$

where T_{new} is the new CCT that will be used in a new transaction, T_{cur} is the CCT that has been used in the previous transaction, S is the secret, and $\|$ denotes concatenation. After the new transaction, T_{new} is stored in the card in place of T_{cur} (in order to perform the next transaction).

C	Actual credit card number
T	One-time credit card transaction number (CCT)
T_{cur}	CCT that has been used in the previous customer transaction
T_{new}	CCT that will be used in the new customer transaction
S	Card-resident secret
Q	Card issuer's verification queue
n	Extending limit
m	Blocking limit

Fig. 1. Notation.

On the customer side, a series of CCTs is generated in continual credit card transactions. On the card issuer side, the same series can be computed and verified. We discuss this in details in Section 3.

2.2. Smart card reader

We need a *smart card reader* to perform the hash computation and update CCT for new transaction. For this purpose, a processing chip is embedded in credit card, and a smart card reader is needed to provide electric power to perform the computation and to update CCTs. Smart card readers are public facilities, which should be available either on shopping sites, or in customers' hands. To encourage the use of new technology, card issuers may give away free smart card readers to customers. Smart card readers can also be integrated into some popular devices such as PDAs, laptops, keyboards and cell phones.

2.3. Payment scenarios

We now present customer payment scheme in different payment scenarios. Assume that smart card readers are always available either on merchant sites or in customers' hands. The discussion on the payment without smart card readers will be given in Section 5.

First consider *on-site payment* scenario where a customer performs a transaction on a merchant's site (e.g., at a food store). In this case, the merchant must have smart card readers available. The customer only needs to insert his credit card into a smart card reader so as to process a transaction. A new CCT is generated and transmitted to corresponding card issuer for verification. Once the transaction is verified, the merchant is notified by the card issuer and the on-site smart card reader writes back the new CCT in place of the old one in the physical card.

Then consider *web payment* scenario where a customer uses his credit card at an e-commerce web site (e.g., at Amazon.com). The customer first gets his CCT updated using a smart card reader, which is connected to his computer. Then the new CCT and other information are transmitted directly to the e-commerce web site (in most cases by SSL). In the case that the smart card reader is not connected to the computer, the customer needs to read out the new CCT from the smart card reader and type it into the e-commerce interface on internet.

The last category of payment scenarios includes *phone payment, fax payment, and email payment* where credit card information is given out by telephones, faxes, and emails (e.g., for hotel reservation). With a smart card reader in hand, the customer can easily update his CCT and use it the same way as in traditional credit card payment.

3. Verification scheme

In this section, we explain how a card issuer verifies a series of CCTs used in credit card transactions. By credit card transaction we mean that a customer sends a CCT to a merchant and vice versa, and by verification a merchant sends a CCT to a card issuer and vice versa. The verification scenarios can be classified into two categories:

- *Instant verification*: a merchant verifies CCT instantly (e.g., on-site shopping).
- *Delayed verification*: a merchant delays the verification of the CCT for a certain period of time (e.g., hotel reservation).

Because of the existence of delayed verifications, the order of CCTs in verification is not necessarily the same as the order in credit card transactions. The following example illustrates our verification scheme.

Example 1. Consider four CCTs T_0, T_1, T_2, T_3 ordered by the time when corresponding transactions are processed, where $T_1 = \mathscr{H}(T_0\|S)$, $T_2 = \mathscr{H}(T_1\|S)$ and $T_3 = \mathscr{H}(T_2\|S)$. Assume that T_0 has been verified by the card issuer most recently.

If all three transactions T_1, T_2 and T_3 are in instant verification scenario, then the CCTs that arrive for verification have the same order as that of their transaction times. The card issuer simply computes the hash chain and verifies them one by one.

```
Verification Algorithm
// I. Arrived credit card information includes an actual credit number $C$ and a CCT $T$.
// II. $C$ is used as an index to find the corresponding customer.
// III. The following only shows the verification of CCT for a particular customer.
// IV. The card issuer knows the secret $S$ and the initial CCT $T_0$.
// V. Two parameters: extending limit $n$ (see line 6) and blocking limit $m$ (see line 12).

1)  current CCT $T_{cur} = T_0$    // $T_0$ is the initial CCT
2)  verification queue $Q = \emptyset$    // initiate a queue for a particular customer

3)  foreach arrived CCT $T$ that is to be verified do
4)      if $T$ matches one CCT in $Q$ then delete it from $Q$ and return CCT verified
5)      else    // if $T$ does not match any CCT in $Q$
6)          generate $n$ new CCTs $T_1, \dots T_n$ where $T_1 = \mathcal{H}(T_{cur}||S), \dots T_n = \mathcal{H}(T_{n-1}||S)$
            // $n$ is a parameter called extending limit
7)          if $T$ matches $T_k$ $(1 \le k \le n)$
8)              $T_{cur} \leftarrow T_k$
9)              insert CCTs $T_1, \dots, T_{k-1}$ into $Q$
10)             return CCT verified
11)         else return CCT not verified    // $T$ does not match any in $T_i$
12)         verification process is blocked if CCTs are not verified $m$ times during certain period of time
            // $m$ is a parameter called blocking limit
```

Fig. 2. Verification algorithm.

Now assume that T_1 and T_2 are in delayed verification scenario and that the CCTs arrive in the order of (T_3, T_2, T_1) for verification. Our scheme verifies them by the following procedure:

1. When T_3 arrives, the card issuer compares it with T_1, which is computed by $\mathscr{H}(T_0\|S)$. Because they do not match, a *verification queue* Q is used to store T_1 for future verification. The card issuer then computes $T_2 = \mathscr{H}(T_1\|S)$ from T_1 and compares it with the arrived CCT. Because they do not match either, T_2 is also inserted into Q for future verification. Finally, the card issuer computes $T_3 = \mathscr{H}(T_2\|S)$ from T_2 and it matches the arrived CCT.
2. When T_2 arrives, the card issuer compares it with the CCTs in the verification queue Q, which consists of (T_2, T_1) at this time. The arrived CCT matches T_2 in Q and thus verified. T_2 is then deleted from Q.
3. When T_1 arrives, it is verified the same way as T_2.

3.1. Verification algorithm

Fig. 2 presents a verification algorithm that verifies customer's CCTs in transactions. At the beginning (lines 1 and 2 in Fig. 2), the current CCT is the initial CCT and the verification queue is empty. The initial CCT is embedded in credit card which is issued to customer. Both card issuer and customer possess an initial

CCT, the actual credit card number, and the shared secret. When a customer initiates a new transaction, a new CCT is generated from the previous one and the secret. The new CCT and the actual credit card number are sent to the card issuer for verification. The actual credit card number serves as an index for the involving customer. Note that the algorithm only presents the verification procedure for a single customer.

The card issuer verifies CCTs one by one. If a CCT is in delayed verification, a later CCT is verified first and the delayed CCT is put into verification queue. Later, the delayed CCT gets verified by matching one of the CCTs in the queue (line 4); the matched CCT is then deleted from the queue.[1] If a CCT is verified before any of its "later" CCTs, it will not be found in the queue. In such case, the card issuer generates n "future" CCTs starting from the current one and checks if the arrived CCT matches one of them (lines 5–11). We call parameter *n extending limit*.

If the arrived CCT matches one of the "future" CCTs starting from the current one (lines 7–10), the matched CCT is used to replace the current one. Those "future" CCTs that are generated before the matched one are put into the queue for future verification.

If the arrived CCT does not match any of the n "future" CCTs (lines 11 and 12), the algorithm returns "CCT not verified". If some CCTs are not verified m times during certain period of time, the verification process is blocked. We call parameter *m blocking limit*.

The use of the two parameters n and m is to thwart certain types of attacks and to tolerate delays and errors in CCT verification. On the one hand, if n is infinite, any random CCT will be verified, leading to no security. On the other hand, if $n = 1$, the scheme tolerates no delays in CCT verification. Similarly, if m is infinite, the scheme permits infinite times of tries of CCTs from an attacker, eventually leading to success of a random try. In the case of $m = 1$, the scheme tolerates no errors (i.e., no retries) in CCT verification.

The selection of extending limit n should be careful. The reason is that a small n may lead to false denial of true CCTs. Let us return back to Example 1. If $n = 2$ and the CCTs arrive in the order of (T_3, T_2, T_1) for verification, then T_3 will not be verified because T_3 matches neither $T_1 = \mathscr{H}(T_0\|S)$ nor $T_2 = \mathscr{H}(T_1\|S)$. In other words, T_3 arrives too "early" compared with the most recently verified T_0. Fortunately, the false denial rate can be made extremely low (to zero) with not-so-large n (e.g., $n = 15$ is large enough to yield zero false denial rate in our simulation). Please refer to Section 3.3 for details.

In reality, most of the delayed payments seldom take more than several weeks, and the majority of the payments are instant payments rather than delayed ones. Consequently, the average length of the verification queue is small (see Section 3.3). However, *queue overflow attack* may occur in some extreme cases. For example, a spoofed merchant who cooperates with a malicious customer may repetitively send a card issuer the nth CCT from the current one; after some time, the queue becomes too long for the limited computing sources. To thwart such attack, a card issuer is suggested to implement one of the following *queueing policies*: (i) control the maximal length of verification queue (i.e., put size constraint); (ii) delete CCTs from the queue that are too old (i.e., put time constraint); (iii) block the queues that increase too fast (i.e., put speed constraint); (iv) combine the above policies. The queueing policies should be implemented in such a way that the normal verification process is not affected significantly.

3.2. The length of verification queue

One can compute the maximal length and the average length of verification queue from a given sequence of credit card transactions. If there is no delay payment, the verification queue will be always empty, otherwise delayed CCTs are inserted into the queue for later verification. The following example illustrates the relationship between the length of queue and the delayed payments.

[1] Besides the queue, both the card issuer and the merchant may need to store some recent transactions for issuing statements or answering inquiries.

Example 2. Consider ten sequential CCTs $T_0, \ldots, T_9$ or- dered by the time when corresponding transactions are processed. Assume that T_5, T_2, T_8, T_1, T_4, T_9, T_6, T_7, T_0, T_3 are the order in which these CCTs are verified and that $t_0, \ldots, t_9$ are the time when these CCTs are verified.

Let q_i denote the length of the verification queue when a payment is verified at time t_i. According to our verification scheme, the length of verification queue changes in the following manner (see Fig. 3).

1. Before T_5 is verified at time t_0, five delayed CCTs $T_0, \ldots, T_4$ are put into the verification queue; that is, $q_0 = 5$.
2. When a delayed T_2 is verified at time $t = 1$, it is eliminated from the queue; the length of the queue is $q_1 = q_0 - 1 = 4$.
3. Before T_8 is verified at time $t = 2$, two more delayed CCTs T_6 and T_7 are put into the queue, leading to $q_2 = q_1 + 2 = 6$.
4. When the delayed CCTs T_1 and T_4 are verified at time $t = 3$ and $t = 4$, respectively, the length of the queue changes to $q_3 = q_2 - 1 = 5$ and then $q_4 = q_3 - 1 = 4$.
5. At time $t = 5$, T_9 is not delayed but next to T_8 which is the latest CCT that has been verified. Therefore, the queue does not change (i.e., $q_5 = q_4 = 4$).
6. After T_9 is verified, the rest CCTs T_6, T_7, T_0, and T_3 are all in delayed verification scenario. Therefore, the length of verification queue decreases to $q_6 = 3$, $q_7 = 2$, $q_8 = 1$, and $q_9 = 0$.

Time	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
CCT verified	T_5	T_2	T_8	T_1	T_4	T_9	T_6	T_7	T_0	T_3
Queue's length	5	4	6	5	4	4	3	2	1	0

Fig. 3. Verification queue in Example 2.

The maximum length of the queue is $\max\{q_0, \ldots, q_9\} = q_2 = 6$. The average length can be computed by $\frac{\sum_{i=0}^{8} q_i \cdot (t_{i+1} - t_i)}{t_9 - t_0}$. If $t_i = i$, the average length is $\frac{\sum_{i=0}^{8} q_i}{9} = \frac{34}{9} \approx 4$.

From the above example, one can generalize the following rules regarding the length of verification queue.

- At any moment, the length of verification queue is the number of delayed payments that have not be verified.
- If a delayed CCT arrives, it decreases the queue length by one. A delayed CCT is a CCT whose transaction time is earlier than one of the CCTs that have been verified.
- When a non-delayed CCT is verified, it increases the queue by the gap (i.e., number of CCTs) between the transaction time of the latest CCT and the transaction time of the non-delayed CCT.

3.3. System simulation

A simulation is implemented to examine the relationship between the verification queue and the combination of the following factors: (i) the model for payment; (ii) the model for verification; and (iii) extending limit n. We examine the average length of the verification queue and the false denial rate in different cases. Note that the blocking limit m has no effect on the verification queue.

In our simulation, a customer's payment is modelled by Poisson process; that is, the time between two payment transactions is exponentially distributed. We use μ_p to denote the mean of exponential distribution, which is the average time

Parameter	Meaning	Default value
μ_p	the average time between two credit card payment transactions	6 hours
μ_v	the average delayed time in the delayed payment scenario	24 hours
r	the fraction of the delayed verifications in all transactions	30%
n	extending limit	10

Fig. 4. Parameters in our simulation.

between two credit card payment transactions by the same customer. A delayed verification is also modelled by Poisson process; that is, the time between a transaction time and its verification time is exponentially distributed. We use μ_v to denote the average delayed time. For each customer, we use r to denote the fraction of the delayed verifications in all transactions.

Fig. 4 lists the parameters and their default values in our simulation. We compute the average length of verification queue during 1000 days of credit card use for each customer. The time period of 1000 days is long enough such that the average length of the queue for a single customer is typical for all customers. In the default case, the fraction of the delayed verifications is 30% ($r = 0.3$); four transactions are processed on average by a customer each day ($\mu_p = 6$ h); the average delayed time for delayed verifications is 24 h ($\mu_v = 24$ h); and the extending limit is ten ($n = 10$). We conduct individual sets of experiments by varying these parameters from their default values.

Fig. 5 shows the average queue length with different average times between transactions. We see that the longer the time period between transactions, the shorter the length of the verification queue. The reason is that, with larger μ_p, fewer delayed verifications are inserted into the queue over a fixed period of time. On the other hand, with longer delayed time in verification, more delayed verifications are inserted into the queue. This results in longer verification queues as shown in Fig. 6. In both figures, the longest queue length is about ten; in other cases, the average length of the queue is less than four.

The ratio of delayed verifications in total transactions has light influence on the queue length as shown in Fig. 7. In particular, a small number of delayed verifications ($r < 0.5$) change the order of credit card transactions in a similar way as a large number of delayed verifications ($r > 0.5$) do. The rough reason behind this trend is that if most verifications are in delayed scenarios, by chance many pairs of them are still in the same order as their transaction times. This can be illustrated by an extreme example where all verifications are delayed with the same period of time and the queue length will be always zero. A theoretic analysis on this remains an interesting topic for further study.

To prevent online-guess attacks, we use the extending limit to restrict the extension of the verification queue. Recall that in our algorithm, if a transaction arrives earlier than some of its previous transactions, the verification queue will be extended and the CCTs for those previous transactions will be put into the queue for future verification. The extending limit is used as a system parameter to control how many transactions a CCT can skip, and this is important in security analysis (see Section 4). Basically, the smaller the extending limit, the harder for an attacker to try a forged CCT,

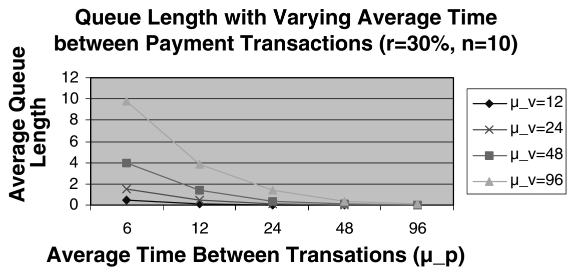

Fig. 5. Queue length with varying average time between payment transactions.

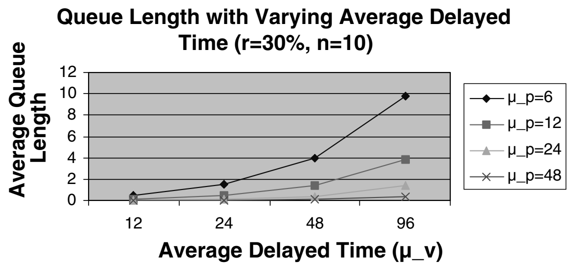

Fig. 6. Queue length with varying average delayed time.

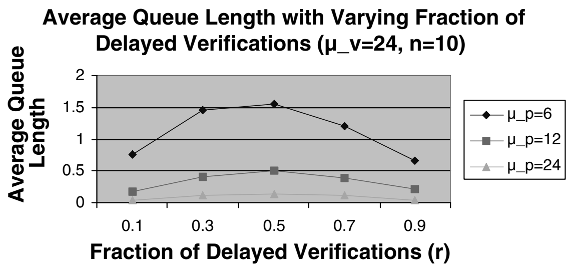

Fig. 7. Average queue length with varying fraction of delayed verifications.

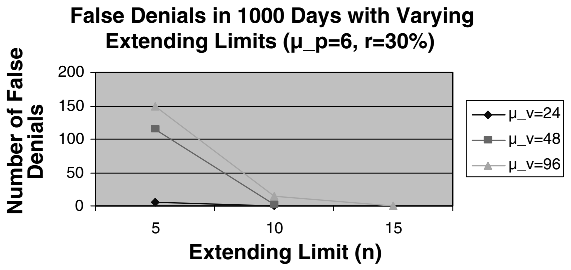

Fig. 8. False alarms with varying extension limits.

and the securer the system. However, the side effect is that if a valid transaction arrives too "early" (i.e., too many of its previous transactions are verified late), its verification may require an extension beyond the extending limit, which leads to false denial. In our simulation, we investigate the number of false denials during 1000 days of credit card use. Fig. 8 shows that the total number of false denials drops dramatically with slightly larger extending limits and that an extending limit of fifteen is large enough to yield zero false denial rate in all cases.

In practice, tradeoff can be made between the false denial rate and the security when selecting the extending limit. For example, the extending limit can be customer-specific – different extending limits are used for different customers based on customer profiles or payment history. The extending limit can also be multi-level or dynamic – the extending limit are tuned to different levels to guarantee low false denial rates and ensure certain levels of security.

3.4. Complexity

We compare the time and space complexity of our verification algorithm with traditional credit card payment. In traditional credit card payment, a customer uses a fixed credit card number for all transactions and the card issuer verifies a payment by checking that number. If the verification queue is used in such case, the length of the queue will be always zero.

Let L be the average length of verification queue in our scheme. On average, our algorithm requires L comparisons between an arrived CCT and those in the queue; an additional comparison is needed between the CCT and the hash value of the current CCT. The time and space complexity of our algorithm is $L + 1$ times of that in traditional payment. Because the average length of the queue is not large (in our simulation, the average length of the queue is close to zero in most cases), the increment of space and time complexity is limited. Such limited increment is affordable considering the fast development of computer hardware described by Moore's law.

The trade-in of the complexity is the security enhancement compared with traditional scheme. Most credit card frauds are can be thwarted due to the use of CCTs. On the other hand, our scheme is convenient for use compared with other one-time credit card number solutions. CCTs can be generated easily on the customer side, and the current protocols such as SSL for transmission of credit card information can still be applied.

4. Security analysis

In our scheme, customers do not need to worry about their actual credit card numbers or identifying information being learned by an attacker because such information is useless in payment without one-time CCTs. In the following, we investigate the security issues when some CCTs are known by an attacker. Unless otherwise stated, we assume that the attacker has no access to a physical credit card nor the card-resident secret (Section 5 will discuss an implementation option in case of card loss).

First assume that the attacker knows a single CCT. Since the attacker does not know the secret, he may choose a random secret and compute a CCT from the known one. The probability that the computed CCT T can be verified is

$$\max\left(\frac{(|Q|+n)}{2^{|S|}}, \frac{(|Q|+n)}{10^{|T|}}\right),$$

where $|\cdot|$ denotes the length of Q, S, or T. The length of Q is constrained by a queueing policy. In default we assume that S is in binary form and T in digits. Recall that our algorithm permits at most m CCTs to be tried in verification; the probability of success of this attack is

$$\max\left(\frac{m(|Q|+n)}{2^{|S|}},\frac{m(|Q|+n)}{10^{|T|}}\right).$$

This probability can be made exponentially low by increasing the length of the secret and/or the CCT. Also note that even if the attacker gets a valid CCT somehow (e.g., purely by chance), it is still useless unless that CCT has not been used in legal transactions by the time the attacker gets it.

Another scenario is that the attacker knows more than one CCTs. Assume that the attacker knows two consecutive CCTs T_1 and T_2. The attacker attempts to know secret S by trying all possible values until the following is attained

$$T_2 = \mathscr{H}(T_1\|S). \tag{2}$$

The average number of tries (until Eq. (2) is attained) is:

$$\frac{2^{|S|}+1}{2}\cdot\min\left(1,\frac{10^{|T|}}{2^{|S|}}\right).$$

If $2^{|S|}\geqslant 10^{|T|}$, the average number of tries is between $\frac{10^{|T|}}{2}$ and $\frac{10^{|T|}+1}{2}$; otherwise the average number of tries is $\frac{2^{|S|}+1}{2}$. This attack can be thwarted by selecting long secret and/or CCTs (e.g., $|S| = 128$ bits and $|T| = 16$ digits are good enough in most cases) such that attaining Eq. (2) is computationally difficult. In addition, even if the attacker finds a secret such that Eq. (2) is attained, the probability that this secret is the customer's real secret is

$$\min\left(1,\frac{10^{|T|}}{2^{|S|}}\right).$$

Again this probability can be made exponentially low by selecting long secret. Another even-if is that the attacker may get the correct secret somehow, it is still useless unless the attacker obtains the current CCT or a not-so-old one in order to compute a valid CCT that can be verified. Since our algorithm permits at most $m(n+|Q|)$ tries, the attacker must obtain a CCT that is no older than $m(n+|Q|)$ from the current CCT if the attacker tries CCTs sequentially.

Now assume that the attacker knows M customers' actual credit card numbers and possibly their identifying information as described in database stealing scenario. With those information, the attacker tries a random CCT for each customer and hopes at least one try would succeed. The probability of success of this attack is

$$\frac{M\cdot(n+|Q|)}{10^{|T|}}.$$

If the attacker tries m CCTs for each customer (before gets blocked), the probability is

$$\frac{M\cdot m(n+|Q|)}{10^{|T|}}.$$

This attack can be thwarted by selecting long CCTs such that the probability is low unless M is extraordinarily large. In the case that the attacker manipulates to know a large number of actual credit card numbers such that one try would succeed, it may take too long time because on average the attacker has to try $\frac{M+1}{2}$ customer accounts (and for each account try m CCTs) in order to find the "right" one.

Finally, assume that the attacker obtains a valid CCT somehow and gets the CCT verified in an illegal transaction. The victim customer will be aware of the credit card fraud as soon as one of his legal payments is abnormally refused or one of his delayed payments cannot be verified (no need to wait for credit card statement as in traditional payment). The refused legal payment will be at most n transactions away from the verified illegal payment due to the extending limit. Early awareness of credit card fraud on the customer side will help reduce the loss of money.

In terms of security, our scheme, like anything else, is not perfect.[2] Particularly, a successful denial of service attack on a card issuer's computing system is ruinous, but this is true for any one-time payment systems, even for the traditional payment. Due to this threat, card issuers' systems are very well protected nowadays.

Due to transmission error or network disconnection, a CCT may not reach corresponding merchant site or card issuer site in online payment scenario. In this case the merchant does not receive any transaction request, or cannot verify the CCT

[2] According to [8], perfect security is not realistic for business. "Good enough [security] is good enough."

with the card issuer. Consequently, the transaction is not confirmed. The smart card reader on customer side thus discard the CCT and drop the transaction request. This will not influence future transactions.

Another possible attack is *replay attack*, where an attacker intercepts a CCT during transmission and quickly sends it to a merchant, hoping that the intercepted CCT can be verified by the card issuer before the legal one. Such attack can be thwarted by using SSL in transmission. In addition, our system is securer than traditional payment since the intercepted CCT is not always useful, but only in the case that the legal CCT is verified later than the intercepted one.

There are other attacks that can be mounted from smart card readers by attackers, especially when we assume that smart card readers have write capability. A malicious operation from smart card readers may release CCTs or other secret information stored in smart chips, it may also results in denial of service attacks by generating false CCTs which cannot be verified by card issuers. In this paper, we focus on the possible attacks that happen during credit card transactions. The security of smart card readers is out of the range of this paper.

5. Implementation options and discussions

In this section, we discuss several implementation options and compare our solution with PKI-based schemes.

5.1. Payment without smart card reader

In some payment scenarios such as phone payment and fax payment, a customer may not have access to a smart card reader. One option is to allow use of actual credit card numbers in this case. This is also the option that is adopted by American Express Private Payments and some other one-time payment systems. Overall such payment system is "mixed" because both one-time numbers and actual credit card numbers are allowed to use (probably in different payment scenarios).

The rational behind the "mixed" system is that the vulnerability of using credit card is different in different payment scenarios. For example, phone payment may be considered "safer" than on-line payment. Actual credit card numbers are allowed to be used in those "safer" scenarios to facilitate transactions.

Though a "mixed" system is convenient to use and it is inter-operable with traditional credit card payment facilitates, it compromises security. Once an attacker learns a customer's actual credit card number (e.g., through shoulder surfing, dumpster diving, or database stealing), the attacker can use it repetitively in permitted scenarios.

Another choice sticks to pure one-time system, in which one-time CCTs are always used in all payment scenarios. Whenever a smart card reader is inaccessible, one must contact with the card issuer (e.g., by phones, emails, or on-line registrations) to get a special CCT. The special CCT is not generated from the previous one as in other cases since the card issuer does not know the last CCT that the customer has used (the last CCT may have not been verified yet). It is also impractical for the customer to remember the last CCT and those special ones.

We propose to generate special CCT T_{spc} from time stamp t_{p} and secret S using a cryptographic hash function $\mathscr{H}$:

$$T_{\text{spc}} = \mathscr{H}(t_{\text{p}} \| S), \tag{3}$$

where t_{p} is the time when the CCT is required by the customer. After generation, the card issuer delivers the special CCT to the customer and inserts it into the verification queue such that it can be verified the same way as normal CCTs. No matter how many times special CCTs are required, it does not affect at all the transactions with normal CCTs. The influence of special CCTs on our verification algorithm is equivalent to that of normal CCTs in delayed verification scenario.

We mention that in our scheme, a customer does not need to reveal his identity to a merchant. An actual credit card number can be used instead of the customer's identifying information in each transaction and serves as the customer's index in verification. However, this is not the case in a "mixed" system since the actual credit card number may be learned by an attacker and then misused in some application scenarios.

5.2. Using personal identification number

So far the security of our scheme is based on the assumption that a physical credit card is not lost. In the case of card loss or theft, the card issuer should be notified in order to prevent credit card fraud. However, it could be too late in some cases.

An option to enhance the security in this case is to use *personal identification number* (PIN) in credit card payment. The PIN number is known by both the customer and the card issuer. In each transaction, the customer punches in his PIN and updates his CCT by hashing the previous CCT, the secret, and the PIN. The card issuer also updates CCTs with the same PIN. In the case of card loss or theft, the blocking limit prevents an attacker from trying PIN too many times. The use of PIN creates a similar security as ATM card in the system of automatic teller machines.

Change of PIN must be done through an authenticated interaction between the customer and the card issuer. During the interaction, the customer informs his new PIN and the last CCT generated with the old PIN. The last CCT is stored in the customer's credit card; it can be read out by a smart card reader and transmitted to the card issuer. Before the card issuer replaces the old PIN will the new one, the gap (in the verification queue) between the current CCT (see Fig. 2) and the last CCT (indicated by the customer) is filled with the CCTs computed with the old PIN.

5.3. Recurring payment

To encourage convenient purchasing or subscription, some e-commerce services (e.g., MSN Wallet) allow a customer to choose *recurring payment*, in which the merchants store the customer's credit card information and repetitively use the same credit card number in several payments that may be split from a single transaction. A pure one-time system requires that the merchants contact with the customer each time in recurring payment to get an updated CCT. While this is secure, it may not be very convenient.

Another option allows some one-time CCTs to be reused under some carefully controlled conditions. Without messing up the verification queue, we propose that such reusable CCTs be generated by Eq. (3) through an interaction between the customer and the card issuer. During the interaction, the customer also indicates the conditions for reusing the CCTs. These conditions may include the number of reusable times, temporal and monetary restrictions, and the corresponding merchants. Such reusable CCTs are specially handled in verification such that the transactions with normal CCTs are not affected.

5.4. Comparison with PKI-based schemes

Public key infrastructure (PKI) can be deployed in various ways for securing credit card transactions. Two possible solutions are SET and SSL, which have been discussed in Section 1.2. In SET, PKI is needed for mutual authentication of involving entities. The dual signatures of order and payment information in SET require that each entity (customer, merchant, credit card issuer or bank) have a public key certificate. This requirement limits the scalability of system as each involving entity must obtain a public key certificate before or during credit card transactions. Because of this limitation, plus the high complexity of the protocol, SET never succeeded in marketplace.

PKI is also used in SSL. Similar to SET, SSL applies PKI for mutual authentication of involving entities before creating a secure communication channel between customer and server (i.e., merchant or card issuer). The 2-way authentication protocol in SSL is seldom deployed because digital certificates on customer side have not been widely used. As a result, most SSL applications enforce server authentication only, though this one-way authentication approach opens a door to potential attacks.

Note that our approach does not exclude the possibility of using SSL in credit card transactions. Since our approach is independent of the communication mode between customers and merchants, we can use SSL in the online payment scenario so as to construct secure communication channels. The only difference is that we use one-time CCTs, instead of real credit card numbers, during the communications.

In summary, the main disadvantage of PKI-based schemes is that each customer is required to possess a public key certificate. This requirement may not be practical for credit card payment. Another disadvantage is that the maintenance of PKI-based payment is not easy due to some management overheads such as certificate revocation, mutual trust of CA's, and etc. Our approach is beyond the existing solutions not only because it applies hash functions, but also supports one-time payment in both online and off-line scenarios, at the same time preserving the features of ease of deployment and maintenance. Our analysis and simulation show that the credit payment, including delayed verification, is secure and practical.

6. Conclusion

We have presented a security enhancement scheme for one-time credit card payment. In this scheme, one-time transaction numbers are generated by hashing their previous transaction numbers and a shared secret between card holder and issuer. The scheme is applicable in both on-line and off-line payment scenarios. Analysis and simulation show that the complexity of our scheme is comparable to that of traditional credit card payment and that the security of scheme is much stronger. We have also discussed several implementation options and compared our scheme with PKI-based approach. We concluded that our scheme is practical for thwarting credit card frauds with a good balance of ease of deployment for credit card companies and ease of use for individual customers.

References

[1] Editorial. Security is in the smart cards. In: eWeek, March 3, 2003, p. 30.
[2] Internet Fraud Statistics Reports. Available from: http://www.fraud.org/internet/intstat.htm.
[3] A.O. Freier, P. Karlton and P.C. Kocher. The SSL protocol. Available from: http://wp.netscape.com/eng/ssl3/ssl-toc.html.
[4] Payment mechanisms designed for the Internet. Available from: http://ntrg.cs.tcd.ie/mepeirce/Project/oninternet.html.
[5] R. Jaques. Identity theft worse than Iraq war. Available from: http://www.vnunet.com/News/1140291.
[6] Private Payments locked with smart chip. Available from: http://home4.americanexpress.com/blue/privatepayments/splash.asp.
[7] A.D. Rubin and R.N. Wright. Off-line generation of limited-use credit card numbers. In: Proceedings of Financial Cryptography, 2001, pp. 196–209.
[8] R. Sandhu, Good-enough security, IEEE Internet Computing 7 (1) (2003) 66–68.
[9] What is SET? Available from: http://www.setco.org/set.html.
[10] A. Shamir, Secureclick: A web payment system with disposable credit card numbers. In: Proceedings of Financial Cryptography, 2001, pp. 232–242.