A Security-Enhanced One-Time Payment Scheme for Credit Card

Yingjiu Li
School of Information Systems
Singapore Management University
469 Bukit Timah Road, Singapore 259756
yjli@smu.edu.sg

Xinwen Zhang
Lab for Information Security Technology
George Mason University
Fairfax, VA 22030, USA
xzhang6@gmu.edu

Abstract

Traditional credit card payment is not secure as semi-secret credit card numbers are repetitively used. One-time transaction numbers have been recently proposed to enhance the security in credit card payment. Following this idea, we use a hash function in generation of one-time credit card numbers: The next one-time number is computed by hashing the current one-time number with a secret that is known only by card holder and issuer. Compared with related work, our scheme places less burden on credit card issuers, and can be easily deployed in both on-line and off-line payment scenarios.

1. Introduction

Millions of dollars loss each year because of credit card fraud has exposed the security weaknesses in traditional credit card processing system. In such system, customers (i.e., credit card holders) repetitively use fixed credit card numbers in all transactions. Because such numbers are "sticky", it is relatively easy for some attackers to steal them. Some common ways are:

- *Shoulder surfing*: An attacker watches a customer from a nearby location as the customer punches in his credit card number or listens in on the conversation if the customer gives his credit card number over the telephone to a hotel or car rental company.
- *Dumpster diving*: An attacker goes through a customer's garbage cans or a communal dumpster or trash bin to obtain copies of credit card statements or other records that bear the customer's identifying information.
- *Packet intercepting*: An attacker sniffs e-commerce packets during on-line credit card payment. In some cases, the attacker does not need to break down the possibly encrypted on-line payment packets (e.g., over Secure Socket Layer), but fools the customer into thinking that he/she is visiting an intended site but actually the attacker's spoofing site.
- *Database stealing*: To encourage easier purchasing, many merchants (who provide services to customers) choose to store credit card numbers in on-line databases. Recent news reported that attackers broke into merchants' sites and stole databases of millions of credit card numbers [1].

Not only does the credit card fraud cause money loss, but also significant worry among customers (especially about on-line transactions). According to a recent study conducted by Opinion Research Corporation, the danger of digital identity theft stimulates even more worry than the war in Iraq [4].

1.1. Related Work

A variety of secure payment systems have been proposed so far to thwart credit card fraud. For practical use, a system should satisfy a set of criteria include: (i)*Ease of deployment*: A system can be deployed (even scaled up) with few additional requirements on current infrastructure and communication protocols; (ii) *Ease of use*: Customers have no difficulties in all payment scenarios; (iii) *Security*: A system should address the real security concerns in current credit card payment system and overcome customers' psychological fear due to credit card fraud. The security may not be perfect, but should be good enough to be user-friendly and business-driven [7].

Among hundreds of solutions [3], Secure Electronic Transactions (SET) protocol (http://www.setco.org/set.html) was designed to protect credit card information in on-line environment. Unfortunately, SET never succeeded in the marketplace because of its high overhead and PKI (i.e., public key infrastructure) requirement.

A successful and widely used solution is credit card payment over Secure Socket Layer (SSL). SSL [2] provides encryption mechanism as well as server authentication for on

line transactions. While a flawless implementation would thwart packet intercepting and server spoofing, SSL solution has no effect on shoulder surfing, dumpster diving and database stealing.

Recently, the concept of *one-time credit card transaction numbers (CCTs)* (also called disposable numbers, single-use numbers, nonces, or tokens in the literature) has been proposed in design of secure payment systems. CCTs are used once in credit card transactions, thus releasing customer concerns that the numbers might be learned by some attackers.

Most of the existing solutions (e.g., American Express' Private Payments [5] and Shamir's SecureClick [8]) require CCTs be generated on-line, from secure interactions between customers and card issuers, during or shortly before credit card transactions. The card issuers associate the CCTs with the customers in stored databases for verification purpose.

As indicated by Rubin and Wright [6], these solutions augment a credit card transaction with an additional connection between a customer and a card issuer. The customer-card issuer communication must be secured, typically by SSL (otherwise CCTs can be learned by someone in the middle before their use). With a large number of customers connecting to a card issuer (or its server) with SSL simultaneously, the performance of the server will become a bottleneck: customers may have to wait long time for CCTs before or during purchase. Such a centralized solution with SSL does not fit the scalability of credit card service. In general, a server with pure function of collecting credit card numbers from customers is dangerous because it is vulnerable to single point failure, web site spoof, and DNS redirection.

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

To bring a remedy to these problems, Rubin and Wright [6] proposed an off-line scheme for generating CCTs (called limited-use credit card numbers or tokens in their paper), without requiring interactions between customers and card issuers in transactions. In their scheme, a card issuer and a customer share a long-term secret key. Before each transaction, the customer generates a CCT by encrypting a set of possible restrictions that describe the purchase in terms of expense, time, merchant and etc. The customer then sends his CCT and his identifying information to a merchant who in turn sends the CCT to the card issuer 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 releases card issuers from on-line SSL connections in CCT generation; however, it burdens them with decryption process, which is expensive especially when there are many customers and restrictions. For practical use of this solution, a customer must have access to a computer or specific device that has encryption capability. The whole process of generating CCTs takes time and requires well-designed user interface to allow a possibly large number of interesting settings of restrictions to be selected and encrypted. In addition, a transaction time-stamp must be included in the encryption process otherwise a generated CCT may not be unique. In some application scenarios, the requirement on encryption devices may not be affordable.

1.2. Our Solution

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

A small chip (smart chip) is embedded into each credit card for hash computations and for storage of a past CCT. To facilitate credit card transactions, smart card readers are needed (in most scenarios) to empower CCT generation.

To process a transaction, a customer simply inserts his/her chipped card into a smart card reader. A new CCT is computed by the smart chip and transferred (e.g., through SSL) to a merchant who in turn transfers (e.g., through SSL) the CCT to a related card issuer for verification. Once the CCT reaches the card issuer, the CCT is verified if it matches the 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 much less burdens on a card issuer because hash computation is simple, fast and usually much cheaper than encryption and decryption. Also a customer is not required to have access to application support for encrypting transaction times and among many restrictions. From customer's view, the whole process is as convenient as in traditional credit card payment provided that chipped cards and smart card readers are available.

As technology evolves, chipped cards and smart card readers become cheap; thus, they can be made publicly available. As an example, American Express' issues its chipped Blue cards and distributes free smart card readers.

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

1.3. Organization

The rest of the paper is organized as follows. Section 2 describes our customer payment scheme in different payment scenarios. Section 3 presents our verification scheme with a system simulation and complexity analysis. Section 4 analyzes the security of our scheme against various attacks. Section 5 concludes the paper.

2. Customer Payment Scheme

2.1. Credit Card

A traditional credit card consists of a semi-secret *credit card number* and other information such as a name and an expiration date. In our scheme, a credit card consists of two *additional* elements: (i) a secret (i.e., a 1024 bits binary string) that never leaves the physical card and never changes; (ii) a CCT (e.g., a 128 bits binary string) that will be used in a single credit card transaction. In the physical card is embedded a small chip that stores the secret and the CCT.

When the credit card is issued by a card issuer to a customer, an initial CCT is stored in the card (therefore, the card issuer knows the secret and the initial CCT for that customer). For each transaction that the customer processes, a new CCT is computed from its previous CCT and the secret by a cryptographic hash function $\mathcal{H}$ (e.g., SHA or MD5):

$$T_{new} = \mathcal{H}(T_{cur}||S) \quad (1)$$

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).

On the customer side, a series of CCTs are 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 of equation 1 and update CCT for a new transaction. A processing chip is embedded in a credit card for the hash computation; the smart card reader provides electric power to perform the computation and update CCT. The smart card reader is a public facility; it should be available either on the site where a customer's transaction is processed, or in a customer's hand. To encourage the use of new technology, card issuers may give away free smart card readers to customers as in American Express Private Payments system. The smart card readers may also be integrated into some popular devices such as PDAs, laptops, keyboards and cell phones.

2.3. Payment Scenarios

We now present a payment scheme in different payment scenarios. We assume that smart card readers are always available either on merchants' sites or in customers' hands. Due to space limit, we leave out discussions on payment without smart card readers as well as other implementation options such as payment with personal identification number, recurring payment, and payment fitting into existing standard for credit cards.

First consider *on-site payment* scenario where a customer performs a transaction on a merchant's site (e.g., at a food store) where smart card readers are available. The customer only needs to insert his credit card into a smart card reader to process a transaction. A new CCT is generated and then transferred 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[1] 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 (e.g., name and address) are transferred directly to the e-commerce application and web site (in most cases by SSL). In the case that the smart card reader is not connected to the computer, the customer needs read out the new CCT (e.g., in digits) 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.

1 Actually the writing back is performed by the smart chip empowered by the smart card reader.

3. Verification Scheme

In this section, we present 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 the merchant sends the CCT to a related card issuer and vice versa. Different verification scenarios can be classified into two categories:

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

Because of the existence of delayed verifications, the order of CCTs in verification is not 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 $T_1 = \mathcal{H}(T_0||S)$, $T_2 = \mathcal{H}(T_1||S)$ and $T_3 = \mathcal{H}(T_2||S)$. Assume that T_0 has been verified by a card issuer most recently.

If all three transactions T_1, T_2 and T_3 are in instant verification scenario, then the CCTs arrived 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.

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). 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 $\mathcal{H}(T_0||S)$. Because they do not match, a *verification queue* Q is used to store T_1 for future verification. The card issuer then computes $T_2 = \mathcal{H}(T_1||S)$ 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 $T_3 = \mathcal{H}(T_2||S)$ 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

Figure 1 presents a verification algorithm[2] that verifies a customer's CCTs in transactions. At the beginning (lines 1 and 2 in figure 1), the current CCT is the initial CCT and the verification queue is empty. The initial CCT is embedded in the credit card which is issued to the customer. Both the card issuer and the customer possess the initial CCT, the customer's identifying information, and the secret. When the customer processes a new transaction, a new CCT is generated from the previous one and the secret embedded in his card; with some delay or without delay, the new CCT, as well as the customer's identifying information, is sent to a related card issuer for verification. Note that the algorithm in figure 1 only presents the verification procedure for a single customer. In multi-customer case, the card issuer identifies a customer using the customer's identifying information as his/her index. The customer's identifying information could be his/her name and address (as used in traditional web payment), or his/her fixed 16 digits "actual" credit card number. In the latter case, CCTs are used in combination of the "actual" credit card number in transactions.

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 the 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[3]. 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 to 11). We call parameter n *extending limit*.

If the arrived CCT matches one of the "future" CCTs starting from the current one (lines 7 to 10), the matched CCT is used to replace the current one. Those "future" CCTs that are before the matched one in the hash chain are then 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

2 The algorithm is described in a way for clear exposition rather than efficiency.

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

// 1. arrived credit card information includes personal identifying information C and a CCT T;
// 2. C is used as an index to find the corresponding customer;
// 3. the following only shows the verification of CCT for a particular customer;
// 4. the card issuer knows the secret S and the initial CCT T_0;
// 5. 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 $T_1, \ldots T_n$ where $T_1 = \mathcal{H}(T_{cur}||S), \ldots T_n = \mathcal{H}(T_{n-1}||S)$
7) **if** T matches T_k $(1 \leq k \leq n)$
8) $T_{cur} \leftarrow T_k$
9) insert CCTs $T_1, \ldots, T_{k-1}$ 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

Figure 1. Verification Algorithm

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 $T_1 = \mathcal{H}(T_0||S)$ nor $T_2 = \mathcal{H}(T_1||S)$. 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 the following section 3.2 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 following section 3.2). However, *queue overflow attack* may occur in some extreme cases. For example, a spoofed merchant who cooperates with a malicious customer may repetitively send the card issuer the n-th CCT from the current one; after some time, the queue becomes too long for the limited computing sources. To thwart such attack, the 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 or affected little.

3.2. System Simulation

A simulation is implemented to examine the relationship between a 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 (i) the average length of a verification queue and (ii) false denial rate (i.e., the probability that a legitimate CCT is not verified) 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 the exponential distribution, which is the average time 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.

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

Figure 2. Parameters in our simulation.

Figure 2 lists the parameters and their default values in our simulation. We compute the average length of the verification queue during 1000 days of credit card use for each

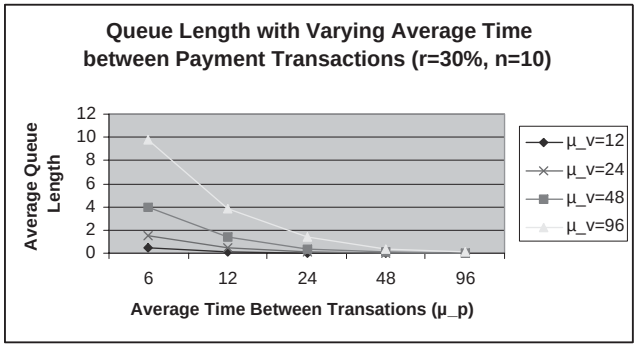

Figure 3. Queue Length with Varying Average Time between Payment Transactions

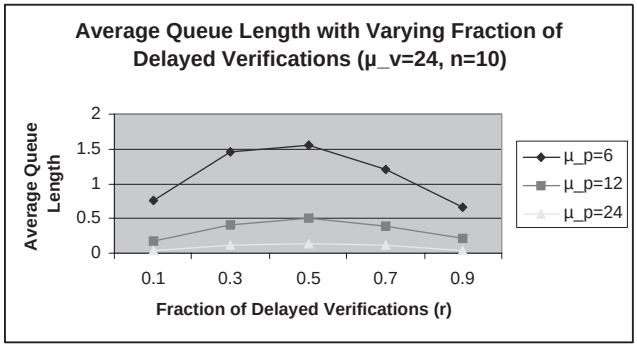

Figure 5. Average Queue Length with Varying Fraction of Delayed Verifications

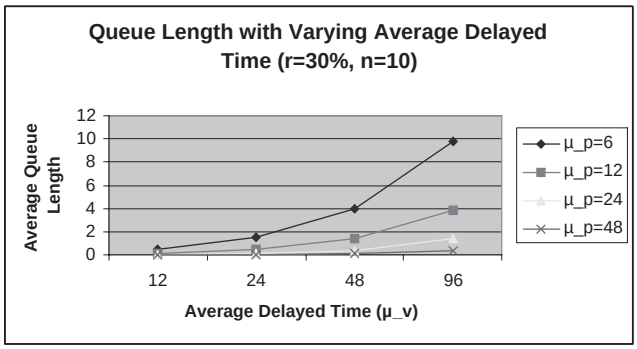

Figure 4. Queue Length with Varying Average Delayed Time

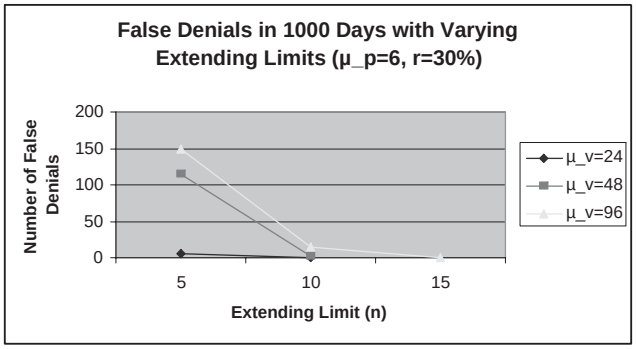

Figure 6. False Alarms with Varying Extension Limits

customer. The time period of 1000 days for credit card use 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 ($\mu_p = 6$ hours); the average delayed time for delayed verifications is 24 hours ($\mu_v = 24$ hours); and the extending limit is ten ($n = 10$). We conduct individual sets of experiments that vary these parameters from their default values.

Figure 3 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 figure 4. In both figures, the longest queue length is about ten in the case of 96 hours of delayed verification and 6 hours of transaction span; 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 Figure 5. Specifically, 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 attacks such as online-guess attacks on one time transaction numbers, 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 extend-

ing limit is used as a system parameter to control how many transactions a CCT can skip, and it is important in security analysis (see section 4). Basically, the smaller the extending limit, the harder for an attacker to try a forged CCT, 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) in an extreme case, its verification may require an extension beyond the extending limit; then a false denial occurs. In our simulation, we investigate the number of false denials during 1000 days of credit card use. In figure 6, we see 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, we may have various ways to tradeoff between false denial rate and security while 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. In terms of security, section 4 shall show that the security is affected linearly by the extending limit while it can be enhanced exponentially with longer CCTs and secrets.

3.3. Complexity

With the average length of the verification queue, we can compare the time and space complexity of our verification algorithm with that in 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 the payment by checking that number. If the verification queue is used in this case, the length of the queue will be always zero.

Let L be the average length of the verification queue in our scheme. On average, our algorithm requires L comparisons between an arrived CCT and those in the queue and an additional one between the CCT and the hash value of the current CCT. Therefore, 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 many cases and less than 10 in the worst case), the increment of space and time complexity in our verification algorithm is limited. Such limited increment is affordable considering the fast development of computer hardware that has been described by Moore's law.

4. Security Analysis

In our scheme, customers do not worry that their "actual" credit card numbers or identifying information may be learned by an attacker because such information is useless in payment without one-time CCTs. In the following, we only 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.

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

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

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

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

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

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

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

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

If $2^{|S|} \geq 10^{|T|}$, 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 $|T| = 10$ digits are good enough in most cases) such that attaining equation 2 is computationally difficult. In addition, even if the attacker finds a secret such that equation 2 is attained, the probability that this secret is the customer's real secret is

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

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[4]. Particularly, a successful denial of service attack on the issuer's computing system is ruinous, but this is true for any other one-time payment systems, even for the traditional payment. Consequently, the card issuer's system should be very well protected, and it has been done so nowadays.

Due to transmission error or network disconnection, a CCT may not reach merchant site and/or card issuer site. In this case the merchant does not receive any transaction request, or he/she cannot verify the CCT with the card issuer. Consequently, the transaction is not confirmed. The smart card reader on the customer side thus discards the CCT and drops current 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 (note that a smart card reader has no write capability even though it empowers a smart chip to compute and update CCTs). A malicious operation from smart card readers may release CCTs or other secret information stored in smart chips; it may also result in denial of service by generating false CCTs which cannot be verified by card issuers. This is a common problem associated with most smart card systems. The security of smart card readers is out of the range of this paper.

5. Conclusion

We have presented a security enhancement scheme for one-time credit card payment that is suitable for practical use. In this scheme, one-time transaction numbers are generated by a hash function from previous transaction numbers and shared secrets between card holders and issuers. The hardware requirements (i.e., smart card readers and physical credit cards embedded with micro chips) can be easily satisfied. The scheme is applicable in both on-line and off-line payment scenarios. Our simulation showed that only small extra burdens are placed on card issuers.

References

[1] Editorial. Security is in the smart cards. In *eWeek*, page 30, March 3, 2003.

[2] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL protocol. *http://wp.netscape.com/eng/ssl3/ssl-toc.html*.

[3] Payment mechanisms designed for the Internet. *http://ntrg.cs.tcd.ie/mepeirce/Project/oninternet.html*.

[4] R. Jaques. Identity theft worse than Iraq war. *http://www.vnunet.com/News/1140291*.

[5] Private Payments locked with smart chip. *http://home4. americanexpress.com/blue/privatepayments/splash.asp*.

[6] A. D. Rubin and R. N. Wright. Off-line generation of limited-use credit card numbers. In *Proceedings of Financial Cryptography*, pages 196–209, 2001.

[7] R. Sandhu. Good-enough security. *IEEE Internet Computing*, 7(1):66–68, 2003.

[8] A. Shamir. Secureclick: A web payment system with disposable credit card numbers. In *Proceedings of Financial Cryptography*, pages 232–242, 2001.

4 According to [7], perfect security is not realistic for business. "Good enough [security] is good enough."