A new Payment Protocol over the Internet

Pierre Girard, Karine Villegas
Gemalto
Avenue du Jujubier – Z.I. Athelia IV
13705 La Ciotat, France
{pierre.girard, karine.villegas}@gemalto.com

Jean-Louis Lanet, Aude Plateaux
XLIM/DMI/SSD
83 rue d'Isle
87000 Limoges, France
{jean-louis.lanet, aude.plateaux}@unilim.fr

***Abstract* — We propose in this paper to reuse the existing payment infrastructure to introduce a proof of transaction genuineness computed by a smart card chip. The idea is to divide the amount of the transaction into several sub-amounts, which added together give the total amount. The sub-amounts are function of a secret shared with the bank, which can verify that the split is correct, thus proving that the transaction is authentic. We provide here a description of the algorithm and its implementation in a .NET card.**

Keywords - payment protocol; credit card transaction; ePayment; authentication; smart card

I. INTRODUCTION

During decades, the usage of magnetic stripes cards was the rule and smart cards the exception. Nowadays, most of the world (with the USA as an exception) has adopted or is about to migrate to EMV [7], the smart card based standard for payment transactions. Paradoxically this technological evolution concerns only payments occurring in the physical world whereas the transactions in the supposedly modern up to date Internet are still stuck with give-me-your-credit-card-number paradigm! Credit card number is also used for transactions over the phone or in paper based (fax, etc) transactions.

A cryptogram[1] printed on card back and composed with three digits in general is also requested to authenticate the card holder. Nevertheless, the security degree added by this additional identification is very low as it is easy to recover by a third party. The card loss or a lack of attention of a card holder who lets all these useful data visible generates an obvious risk. Attacks in vendors systems, by social engineering or by phishing are also routinely used by attackers.

One goal of the present work is to improve the security level of a payment transaction using existing infrastructure such as phone, computers, or fax. Many secure solutions have been proposed (like using personal payment terminal accepting smart cards). However none of them have gained acceptance as they supposed a significant investment. We will leverage on already deployed smart cards and cheap unconnected readers. These readers are sometime deployed by banks to produce One Time Password for home banking. It will be more and more the case as numerous laws now mandate a two-factor authentication for home banking. In any case these small devices are cheap. They are not secure and feature only a numeric key pad and a small LCD display. An example is shown in the Fig. 1.

Figure 1. Example of a simple portable smart card reader

One mean to reach our security objective is to use the fact that it is possible to store in non volatile memory of a banking card some secret information only shared between the card and the bank. The secret information is linked to the card serial number. The bank is able to retrieve the secret key of each card that it has delivered. Currently, the secret key is not used in payment modes using the phone, internet or the fax and in this work we show a way to change that.

We propose to reuse the existing payment infrastructure (which conveys already at least the card number, the transaction amount and the transaction time) to introduce a proof of transaction genuineness computed by the smart card chip. The basic idea is to split the transaction into several sub-transactions, which added together give the same amount. The amounts of the sub-transactions are function of a secret shared with the bank, which can verify that the splitting is correct, thus proving that the transaction is genuine. The number of sub-transactions can be adjusted to the optimal security-performance trade-off.

From a practical point of view, the customer, before paying on the Internet, will insert her smart card in the

[1] Called CVV2 (Card Verification Value 2), this code has a modest but real advantage: as it is not present on the magnetic stripe, it will not be obtained by skimming attacks.

portable reader and enter her PIN. Then, she will enter the amount she's willing to pay (let's say 100 €). As a result the reader will display a small list of numbers which sum is the original amount (for example 1 € + 19 € + 80 € = 100 €). Finally the customer will perform three payments on the Internet (one of 1 €, one of 19 € and one of 80 €). The split of 100 in 1, 19 and 80 is a function of a secret key (contained inside the smart card) and the date. This split is the proof that the transaction has been initiated by a genuine customer owning the smart card and knowing the PIN.

The rest of this paper is organized as follows. In the section 2, we give an overview of the related work. Then in section 3-4, we present the principle of our algorithm and its correctness. In section 5, we evaluate our protocol security. In section 6, a detailed implementation is presented. Finally in section 7, we conclude.

II. RELATED WORK

There are several protocol proposals for e-transaction and most of them require installing plug in or they are not able to guarantee the same security properties than our protocol. We evaluate here the most representative secured payment protocols such as Secure Sockets Layer (SSL), Secure Electronic Transaction (SET) [5], Three Domains Secure Electronic Transaction (3D-SET) and Three Domains Secure (3D-Secure) [1, 6].

The protocol SSL (Secure Socket Layer) was launched in 1994 by Netscape with the goal of providing secure communications between web browsers and web servers. Then this protocol was renamed TLS (Transport Layer Security) in 2001. SSL/TLS protocol provides authentication and encryption communications between clients and servers. Although SSL/TLS was not designed to be a payment system, it appears to be the most common way to do a digital transaction nowadays. When used in an e-commerce transaction the merchant cannot identify the cardholder. If a stolen credit card is used for a transaction, the merchant is responsible for "card not present" transaction charge back. SSL does not protect stored data on the merchant's server. The merchant is allowed to see the payment information. Nothing prevents the merchant from misusing this information.

In 1996, Visa and MasterCard developed the protocol Secure Electronic Transaction (SET). It relies on the utilization of certificates and signatures. It guarantees the integrity and the confidentiality of data during the authenticate each other. The main disadvantage is related to the cryptographic mechanisms, which implies computing resources. The cardholder needs a piece of software and a certificate, installed on his computer. The merchant also has a heavy implementation. The utilization of the SET protocol is not obvious. The cardholder can only order from one computer because of the certificate. This protocol has not been a success. A version using smart card on customer side (named C-SET) was even derived from SET with no more success.

3D-SET is a 3D payment system developed by Visa in 2001. It sets up the SET protocol into the 3D model and reduces the means which have to be deployed at the merchant and customer. In fact, with 3D-SET, there are 'thin' modules for merchants and 'slim' wallets for cardholders. The main disadvantage of this protocol is that for each card owned by the customer, he must have a certificate and a digital wallet. Moreover, 3D-SET is not interoperable with SSL websites.

The 3D secure protocol [2, 3, 4] aims at reducing frauds by strengthening authentication. It's called "Verified by Visa" for Visa and "Secure Code" for MasterCard. It uses XML messages sent over SSL connections with client authentication. It was created in 2001. The disadvantages are: the merchant's server has access to the card number and the expiration date. The merchant can redirect the cardholder to a fake access control server. It is difficult to distinguish between the Verified by Visa pop-up window and a fake site. For a complete security study, see [8].

Thus, SET was too complex and too expensive to implement. It was never really adopted, although he provided a high level of security. 3D-SET brought some changes but neither was it adopted. 3D-Secure and SSL are the two main payment protocols used nowadays. Although it does not guaranty the integrity of data and it does not authenticate the cardholder (that allows frauds), SSL is still widely used in the e-commerce. 3D-Secure provides mutual authentication between the merchant and the cardholder. In fact, authentication of merchant to cardholder is performed by the use of SSL and authentication of cardholder to merchant is performed by the use of the ACS. And for the non-repudiation, the fact that each entity is authenticated before processing the transaction, cardholder and merchant cannot deny having participated in this transaction.

III. THE PRINCIPLE OF THE ALGORITHM

The reader sends to the card the date and time (denoted D) and the transaction amount (denoted A). The card stores a secret key K, shared with the bank. The card uses a function R (taking in argument D, A and K) to obtain a pseudo-random bit stream S. The card chooses a number n according to A (see section 5. Security of the protocol) and uses a function F (taking S and A as arguments) to produce n sub-amounts A_i such that:

$$A = \sum_{i=1}^{n} A_i$$

The card sends the Ai to the reader. The n transactions of Ai amounts with the same date and time are entered in the payment system. For a given card, the bank groups the n transactions with the same date and time D and computes the sum:

$$A = \sum_{i=1}^{n} A_i$$

of the different amounts A_i.

The bank retrieves the secret key K of the card.

The bank computes $S = R(D, K, A)$.

The bank computes $\{A'_0,\ldots,A'_n\} = F(S, A)$.

The bank verifies that for each i, $A_i = A'_i$, if yes the n transactions are authenticated.

A. The R function

For the payment protocol, *R* is a keyed hash function, supposed to be one-way, and collision free, and specifically we could use HMAC-SHA2 (D||A, key) that we can find in Java card API 2.2.2.

The function *R* will return a pseudo-random bit stream *S*.

B. The F function

The function *F* computes n sub-amounts *Ai* and sends one sub-amount over as the card produces one (we do not store them).

This function is used for collecting the result of successive divisions and modular reductions of the bit stream *S*. We determine the value of the concerned *Ai*, and for that we must find its high-order and low-order bit in the bit stream *S*.

IV. CORRECTNESS OF THE ALGORITHM

We defined the algorithm through the previous parts. We are now ready to prove its correctness. For that, we must prove that:

- all operations in the protocol are adequately defined;
- the algorithm comes to an end;
- the algorithm returns the expected result.

Let's first demonstrate that all operations are adequately defined. This is equivalent to prove that all computations made during the execution of F are correctly defined. We use: a pseudo-random bit stream S, returned by a hash function; the transaction amount A; the number of sub-amounts n; k is chosen as the smallest integer such that: $2^k = \frac{A}{n} + \varepsilon$, where ε is a very short positive number.

That is to say, we have: $2^{k-1} < A / n < 2^k$.

We will then compute:

- the sub-amounts $A_i = \frac{S}{2^{(n-i)k}}$: we know *S*, *n* and *k*, and $2^{(n-i)k} \neq 0$, which means that there is no division by zero;
- these previous results modulo 2^k : this is a simple modular reduction and is correctly defined for all A_i;
- and, finally, the last sub-amounts $A_n = A - \sum_{i=1}^{n-1} A_i$: we just compute all A_i for $i \in \{1,\ldots,(n-1)\}$ and we know *A*, which means it is easy to calculate A_n.

Therefore, all operations in the algorithm are adequately defined.

We'll now prove that the algorithm comes to an end. The algorithm uses a hash function to compute S, then calls F to calculate all A_i. It means that the protocol comes to an end if F does. But as defined previously, the A_i represent the splitting of the transaction amount *A*. This amount being finite, the number *n* of A_i is finite. F stops when it has computed *n* A_i : the protocol comes to the end.

Finally, let's prove that the protocol returns the expected result. The algorithm's goal is to compute a splitting of the transaction amount A, but we define the last sub-amount A_n as $A - \sum_{i=1}^{n-1} A_i$. Therefore, we are sure to obtain the expected result: $A = \sum_{i=1}^{n} A_i$.

V. SECURITY OF THE PROTOCOL

Here we demonstrate that the protocol is significant with a negligible probability that means that the probability for an attacker to be mistaken for the smartcard is very small. We'll suppose that the opponent managed to procure *A*, *n* and *D* for a given transaction. His goal will be to re-compute the right splitting of *A* without knowing the key *K* to get the server to accept an illegal transaction. Knowing *A* and *n*, he can easily re-compute *k*. What, then, is the probability for him to find the right splitting?

If the opponent manages to compute the (n-1) first A_i, he easily obtains the last one:

$$A_n = A - \sum_{i=1}^{n-1} A_i .$$

But by definition: $A_i \in \left[\left| \ 0, \frac{A}{n} + \varepsilon \ \right| \right]$ for $i \in \{1,\ldots,(n-1)\}$

it is equivalent to: $A_i \in \left\{0,1,\ldots., \frac{A}{n} + \varepsilon\right\}$.

Now: $\#\left\{0,1,\ldots., \frac{A}{n} + \varepsilon\right\} = \frac{A}{n} + \varepsilon + 1$

Therefore, the opponent has to choose amongst $\frac{A}{n} + \varepsilon + 1$ values for each A_i, which means that the probability he will find the correct A_i is $\frac{1}{\frac{A}{n} + \varepsilon + 1}$.

This means that the probability P of him finding the right splitting for *A* is: $P = \frac{1}{\left(\frac{A}{n} + \varepsilon + 1\right)^{n-1}}$.

In order to guarantee that P will be negligible, we need to lay down some rules on *A* and *n*. Let's consider the security of the splitting of *A*: we'll prove here that P is negligible if *n* and/or *A* is up.

<u>If *n* is up:</u>

Then the value of $\left(\frac{A}{n} + \varepsilon + 1\right)^{n-1}$ is high since $\frac{A}{n} + \varepsilon + 1 > 0.$

Therefore, the probability that an attacker will find the right splitting, which is

$\frac{1}{\left(\frac{A}{n} + \varepsilon + 1\right)^{n-1}}$, is low.

<u>If *n* is low:</u>

For $\frac{1}{\left(\frac{A}{n} + \varepsilon + 1\right)^{n-1}}$ to be a negligible value, *A* needs to be high.

As such, the splitting of *A* is secure if *n* and/or *A* is high.

Let's fix the maximum value of P as 2^{-50}.

We then have the following equivalences:

$$\frac{1}{\left(\frac{A}{n} + \varepsilon + 1\right)^{n-1}} \le 2^{-50} \quad \Leftrightarrow \quad 2^{50} \le \left(\frac{A}{n} + \varepsilon + 1\right)^{n-1}$$

$$\Leftrightarrow \quad \left(\sqrt[(n-1)]{2^{50}} - \varepsilon - 1\right) n \le A$$

$$\Leftrightarrow \quad \left(\sqrt[(n-1)]{2^{50}}\, n - n\right) - \varepsilon . n \le A . \qquad \textbf{(1)}$$

Let's study the evolution of the nth root of a constant c, that is to say $f(n) = \sqrt[n]{c} = c^{1/n}$:

n	37	$+\infty$
$1/n$	$+\infty$	0
$\sqrt[n]{c}$	+	
$f(n)$	$+\infty$ ↘	1

This function is decreasing, and more particularly for the values of *n* that are of concern to us: $n \in]0,...,+\infty]$. Therefore, the higher *n* is, the smaller $\sqrt[n]{c}$ becomes.

Considering the function $\sqrt[n]{c}$ is decreasing on $n \in]0,...,+\infty]$, the function $\sqrt[(n-1)]{2^{50}}$ is decreasing on $n \in]1,...,+\infty]$.

Then, let's study the variation of the function *g(n)* $= \sqrt[(n-1)]{2^{50}}\, n - n$:

Let's compute *g(n+1) – g(n)*:

$$\begin{aligned} g(n+1)-g(n) &= \sqrt[n]{2^{50}}(n+1) - (n+1) - \left(\sqrt[(n-1)]{2^{50}}\, n - n\right) \\ &= \sqrt[n]{2^{50}}(n+1) - 1 - \sqrt[(n-1)]{2^{50}}\, n \\ &= \sqrt[n]{2^{50}}\, n + \sqrt[n]{2^{50}} - 1 - \sqrt[(n-1)]{2^{50}}\, n \\ &= n\left(\sqrt[n]{2^{50}} - \sqrt[(n-1)]{2^{50}}\right) - 1 + \sqrt[n]{2^{50}} \end{aligned}$$

However, the function $\sqrt[(n-1)]{2^{50}}$ is decreasing on $n \in]1,...,+\infty]$.

Consequently, $\sqrt[n]{2^{50}} < \sqrt[(n-1)]{2^{50}}$, therefore $\sqrt[n]{2^{50}} - \sqrt[(n-1)]{2^{50}} < 0$ on $n \in]1,...,+\infty]$.

And so if $n \to \infty$ then $\sqrt[n]{2^{50}} - \sqrt[(n-1)]{2^{50}} \to 0^-$.

Moreover : if $n \to \infty$ then $\sqrt[n]{2^{50}} \to 0$.

Conclusion: if $n \to \infty$ then $n\left(\sqrt[n]{2^{50}} - \sqrt[(n-1)]{2^{50}}\right) - 1 + \sqrt[n]{2^{50}} \to 0^- - 1 + 0 = -1$.

And consequently :

g(n+1) – g(n) < 0 ⇔ g(n+1) < g(n)

And so g(n) = $\sqrt[(n-1)]{2^{50}}\, n - n$ is decreasing on $n \in]1,...,+\infty]$.

As such, the function $\left(\sqrt[(n-1)]{2^{50}}\, n - n\right) - \varepsilon . n$ is decreasing. From the relation **(1)** : $\left(\sqrt[(n-1)]{2^{50}}\, n - n\right) - \varepsilon . n \le A$, we can now deduce that: the

upper n is, the lower $\left(\sqrt[(n-1)]{2^{50}}n-n\right)-\varepsilon.n$ becomes. A can then be relatively small.

We notice that with:

- $n=53$

$$A \geq \left(\sqrt[(n-1)]{2^{50}}n-n\right)-\varepsilon.n$$
$$=\left(\sqrt[(53-1)]{2^{50}}.53-53\right)-53.\varepsilon$$
$$\approx 50{,}211-53\varepsilon$$

- $n=54$

$$A \geq \left(\sqrt[(n-1)]{2^{50}}n-n\right)-\varepsilon.n$$
$$=\left(\sqrt[(54-1)]{2^{50}}.54-54\right)-54.\varepsilon$$
$$\approx 49{,}845-54\varepsilon$$

Therefore if $n \geq 54$ then: $A \geq 49{,}845-54\varepsilon$.

Which means that if $n \geq 54$ we can afford to have $A \geq 50$, since $50 \geq 49{,}845-54\varepsilon$ where $\varepsilon \geq 0$.

Accordingly, we consider in this project that $A \geq 50$, which means that the minimal amount of the transaction is 50 cents. With this assumption, we are then able to choose $n \geq 54$. With these two conditions, we are sure that P is negligible.

VI. IMPLEMENTATION

We developed a complete prototype that includes a server for the verification, a terminal and a smart card. The Central Server, located in a secure location, gives us a maximal protection. It connects the terminals by TCP/IP, which can easily evolve with IPSec, VPN, or another network protection. The terminal, a small embedded device gives us a go-as-you-please portability. It communicates with the cards by PC/SC, a worldwide standard supported by all smart cards (see Fig. 2). The payment card, holding by the client, implements our payment protocol.

Figure 2. : Implementation settings

This set-up was devised for simple prototyping and protocol validation. The final architecture will use a lightweight unconnected reader as shown in section 1. The user will copy manually the reader output in her browser. A connected version along with some auto-form filling plug-in can also be imagined.

The payment protocol advices us to use R as a keyed hash function, which is supposed to be one-way, and collision free, and specifically HMAC-SHA2 (D||A, key). We have chosen to implement the protocol on a .NET card[2], but it should have been possible to implement it on a Java Card 2.2.

The Gemalto .NET Card contains a CIL interpreter that allows users to develop applications for the smart card using the ECMA .NET standard. Applications can be developed in any language that can be compiled to CIL. The runtime environment consists of two components. The first is an interpreter; it is responsible for running applications that are loaded to the card. The second component is a collection of libraries that support applications. On a Java Card, these libraries would contain the types and methods that are part of the Java Card API. On the Gemalto .NET Card, these libraries contain a subset of the ECMA .NET libraries.

The function F computes n sub-amounts A_i and sends one sub-amount over as the card produces one (we do not store them) in using the bit stream S as shown in Fig. 3.

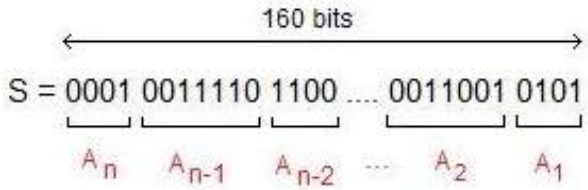

Figure 3. Usage of bit stream S

In the protocol, the function F takes S and A as arguments to produce n sub-amounts A_i, but for practical reasons our implementation takes n and k as arguments too:

private void functionF (byte[ ] S, short i, short k) {…}

S the pseudo-random bit stream returned by the function *R*
i the number of *Ai* that wants the terminal
k the exponent of 2 power upper than *A/n* : $2^k = A/n + e$.

A. *Declaration and initialization*

We determine the index of the byte (*bitFDiv*) where the low-order bit in the bit stream *S* is, and the index of this low-order bit (*bitFMod*) in this byte:
*byte bitFMod=(byte)((i*k)%8)*
*byte bitFDiv=(byte)((i*k)/8).*

B. *Operations*

We begin to store in the variable *result* the high-order bits of *S[bitFDiv]* from *bitFMod* to the end of the byte (with a shift). We have created a mask in order to make sure that in *result* the bits beyond the index *bitFMod* are null:
mask = (short) (0x00FF >> bitFMod)

[2] The source code is available at : http://secinfo.msi.unilim.fr/site/index.php?n=Softwares.Softwares

result = (short) (((short) S[bitFDiv] >> bitFMod) & mask).

Then we determine the number of reminded bits of A_i:
miss = (short) (k - (8 - bitLowMod)).

We store in the variable *reste* the byte where these reminded bits are. We have to test for the case which the bits of Ai are distributed on 3 bytes of S. Then we apply a mask to *reminder* in order to erase the too many bits:
mask = (short) (0x00FF >> (8 - miss))
reminder = (short) ((S[bitLowDiv + 1]) & mask).

We shift *reminder* by the number of bits that we have put in *result*, and we xor the both, so that we obtain finally the wanted *Ai* in *result:*
result ^= (short) (reminder << (8 - bitLowMod)).

The terminal does a loop asking each time to send him the successive A_i. In order to be sure that the terminal has received each sub-amount, the card returns the number *i* of the A_i it needs. So if the card receives a *i* different from the previous one, it means that the terminal has correctly received the precedent *Ai*. We compute then the new *Ai* calling the function *F(S, i, k)*. Then we deduct this *Ai* to the total amount *A*, in order to obtain at the end the final A_N.

If the card receives a number *i* corresponding to the last *Ai* called, then we only re-compute the *Ai* without deducting again its value from *A.* If the card receives an *i* equals to the last *Ai* (*=N-1*), we send the result of all successive subtractions.

VII. CONCLUSION

We proposed a new payment protocol that reuses the existing payment infrastructure. The idea is to split the transaction into several sub-transactions, which added together give the expected amount, with a verification of splitting according to a secret shared with a central server. In addition, the end user can also verify the authenticity of its transactions using its own card as an oracle. We developed an implementation of this proof of concept on a dot net card connected to a server that verifies the correctness of the payment.

REFERENCES

[1] P. Jarupunphol, C.J. Mitchell : Measuring 3-D Secure and 3D-SET against e-commerce end user requirements.

[2] Visa : Verified By Visa, Introduction, December 30, 2006 .

[3] Visa : Verified by Visa, System Overview External Version 1.0.2, December 30, 2006 .

[4] Visa : 3D-Secure Protocol Specification Core Functions, July 16, 2002.

[5] Visa & MasterCard : SET Secure Electronic Transaction Specification, Book 1: Buisiness Description, May 31, 1997 .

[6] Ciprian Stanică-Ezeanu : Comparative Study of Internet Payment Security Protocols Based on Credit Cards, 2008.

[7] EMV, Integrated Circuit card Specifications for Payment Systems, Version 4.2, June 2008. http://www.emvco.com/.

[8] Verified by Visa and MasterCard SecureCode: or, How Not to Design Authentication, Steven J. Murdoch and Ross Anderson, Financial Cryptography and Data Security '10, 25-28 January 2010, Tenerife.

.