

Secure acceleration of DSS signatures using insecure server

Philippe Béguin*

Philippe.Beguin@ens.fr
Laboratoire d'informatique **
Ecole Normale Supérieure
45, rue d'Ulm
F-75 230 Paris Cedex 05
FRANCE

Jean-Jacques Quisquater

Quisquater@dice.ucl.ac.be
Laboratoire DICE
Université Catholique de Louvain
Place du Levant, 3
B-1 348 Louvain-la-Neuve
BELGIUM

Abstract. Small units like chip cards (*smart card*) have the possibility of computing, storing and protecting data. Today such chip cards have limited computing power and some cryptoprotocols are too slow. Some new chip cards with secure coprocessors are coming but are not very reliable at the moment and a little bit expensive.

A possible alternative solution is to use an auxiliary unit in order to help the chip card. The known protocols are not very secure or are not efficient.

We show how to accelerate the computation of $a \times b \bmod c$ and of $a^t \bmod c$ where a, b, c, t are public. Next we show how to accelerate the discrete exponential modulo a prime number: this protocol is useful to accelerate DSS signatures and other schemes. This protocol is also the first one accelerating DSS signatures with the help of an insecure server: it is secure against both passive and active attacks (that is, when the server sends false values to get some information from the card). Moreover, this protocol is the first secure such a protocol which does not use precomputations in the card.

We describe a feasible version of these protocols, where the used RAM is small: with current chip cards it is thus possible to implement effectively such protocols.

1 Introduction

Small units like chip cards (another name is *smart card*) have the possibility of computing, storing and protecting data. Today such chip cards have limited computing power and some protocols are not executed in an efficient way (for instance, some protocols with public key cryptosystems). Some new chip cards with fast and secure coprocessors are coming but are not very reliable at the

* Part of this work was done while the author was visiting the Laboratoire de Microélectronique, Université Catholique de Louvain, Belgium.

** Supported by the Centre National de la Recherche Scientifique URA 1327.

moment (due to problems of auxiliary memory): another weak point is the associated cost.

A possible alternative solution is to use an auxiliary unit (a banking terminal, a card reader, ...) in order to help the chip card. In this paper we shall use the words *card* for the main unit and *server* for the auxiliary unit.

If the server is secure it is possible to imagine a secure link between the card and the server: the card sends the secret values to be used to the server; the server computes the result and sends it using again the secure link. The interesting (real life) case is working with an insecure server. This server may be under the influence of an opponent trying to obtain the secrets of the card or to cheat with a false result. The conclusion of this short analysis is that the card must protect its secrets and verify the computations received from the server.

Such protocols were first studied by Matsumoto, Kato and Imai [6] and Quisquater and De Soete [9]. Next Pfitzmann and Waidner [8] and Anderson [1] have shown that these protocols are not very secure. Then Yen and Lai [12], Matsumoto, Imai, Lai and Yen [5] and Kawamura and Shimbo [4] propose protocols using expensive precomputation and then not very efficient.

In this paper, we show how to accelerate the computation of $a \times b \bmod c$ and of $a^t \bmod c$ where a, b, c, t are public. Next we show how to accelerate the discrete exponential modulo a prime number: this protocol is useful to accelerate DSS signatures [7] and signature schemes based on the Schnorr's scheme [11]. We describe feasible version of these protocols, where the card does not need precomputations and where the used RAM is small: with current chip cards it is thus possible to implement effectively such protocols.

This protocol is secure against passive and active attacks: when the server sends false values to get some information from the card. Thus we can use this protocol with current chip cards to accelerate the DSS signatures without decreasing the security of this signature.

In this paper we denote multiplication $M \times N$ (*resp.* modulo $M \times N$) a multiplication of an M bits number by an N bits number (*resp.* a modulo of an M bits number by an N bits number). We call modular multiplication, a multiplication of two 512 bits numbers modulo a 512 bits number. We denote by $|a|$ the number of bits of a (i.e. $|a| = \lfloor \log_2 a \rfloor + 1$).

We count the number of computations done by the card in number of modular multiplications. Here we consider that the computation of $a \times b$ is about the equivalent of $\frac{1}{2} \times \frac{|a|}{512} \times \frac{|b|}{512}$ modular multiplications and $a \bmod b$ is about the equivalent of $\frac{\theta}{2} \times \frac{|a|}{1024} \times \frac{|b|}{512}$ modular multiplications. By θ we denote a constant near 1 varying with the implementations of the operation modulo. Here we use the practical value 1.25 for θ . In the context of the card we consider as equivalent the modular square and the modular multiplication.

2 Computation of $a \times b \bmod c$

Let a, b, c be non negative integers such that $a < c$ and $b < c$. The card gives a, b, c to the server. The goal is to obtain the result of $a \times b \bmod c$ and a proof that

this result is correct with a high probability: here we hope that the probability for the card of accepting a false value is less than $\frac{1}{2^{20}}$ or $\frac{1}{2^{64}}$. The designer will choose the level of security depending on the context.

The protocol is the following one:

1. The card chooses k secret prime numbers p_1, \dots, p_k such that for all i , $|p_i| = N_v$.
2. The card sends to the server a, b, c .
3. The server computes and sends to the card

$$\alpha = \lfloor (a \times b)/c \rfloor,$$

$$r = (a \times b) \bmod c.$$
4. The card verifies:

$$\alpha < c,$$

$$r < c,$$
 for each number p_i : $(a \times b) \bmod p_i = (\alpha \times c + r) \bmod p_i$.
5. If during this former step the verification is correct, then the card accepts the result: $a \times b \bmod c := r$.

In order to minimize the computations during the verification $(a \bmod p_i) \times (b \bmod p_i) \bmod p_i = (\alpha \bmod p_i) \times (c \bmod p_i) + (r \bmod p_i) \bmod p_i$, the card computes:

1. $\alpha_i = \alpha \bmod p_i$,
2. $c_i = c \bmod p_i$,
3. $e_i = \alpha_i \times c_i$,
4. $a_i = a \bmod p_i$,
5. $b_i = b \bmod p_i$,
6. $d_i = a_i \times b_i$,
7. $f_i = (r + e_i - d_i) \bmod p_i$.

Next the card verifies $f_i = 0$. That is, for each number p_i , the card needs 5 modulo $512 \times N$ (steps 1, 2, 4, 5, 7) and 2 multiplications $N_v \times N_v$ (steps 3 and 6); thus, the card needs the equivalent of $5 \cdot \frac{\theta}{2} \frac{512}{1024} \frac{N_v}{512} + 2 \cdot \frac{1}{2} \frac{N_v}{512} \frac{N_v}{512}$ modular multiplications.

Now we study the possibility for the server of cheating, that is, the card accepts a false value r^* instead of r .

Theorem 2.1 *Let a, b, c be non negative integers such that $|a|, |b|, |c| \leq N$ and $a, b < c$. If the card chooses secretly k random prime numbers of length N_v , then the probability ϱ that the card accepts a false value satisfies*

$$\varrho \leq \frac{\binom{\lfloor \frac{2N}{N_v-1} \rfloor}{k}}{\binom{M}{k}},$$

where M is the number of prime numbers of length N_v .

Proof. Let α, r such that

$$a \times b = \alpha \times c + r \quad \text{where } r < c. \quad (1)$$

Then α, r are uniquely determined and $\alpha < c$.

Suppose the card receives α^*, r^* such that $\alpha^* < c, r^* < c$ and $r^* \neq r$. By the unicity of (1),

$$a \times b \neq \alpha^* \times c + r^* \quad (2)$$

Let p_1, \dots, p_M the prime numbers of length N_v .

Let $I(\alpha^*, r^*) = \{i : (a \times b) \bmod p_i = (\alpha^* \times c + r^*) \bmod p_i\}$.

By the Chinese Remainder Theorem,

$$(a \times b) \bmod P = (\alpha^* \times c + r^*) \bmod P, \quad (3)$$

where $P = \prod_{i \in I} p_i$.

Since $|p_i| = N_v$, then $p_i \geq 2^{N_v-1}$, hence $P \geq 2^{l(N_v-1)}$ where $l = |I|$.

Since $|a|, |b|, |c| \leq N$ and $r^*, \alpha^* < c$, then $(a \times b) < 2^{2N}$ and $\alpha^* \times c + r^* < (2^N - 1) * 2^N + 2^N = 2^{2N}$.

If $2^{l(N_v-1)} \geq 2^{2N}$ then $P > (a \times b)$ and $P > \alpha^* \times c + r^*$. By (3), we have $a \times b = \alpha^* \times c + r^*$, which contradicts (2).

Hence $2^{l(N_v-1)} < 2^{2N}$ then $l \leq \lfloor \frac{2N}{N_v-1} \rfloor$.

The card chooses randomly k prime numbers $\{p_j\}_{j \in J}$ where $|J| = k$. The card accepts r^* if and only if $J \subset I$. Then the probability ϱ that the card accepts r^* satisfies

$$\varrho \leq \frac{\binom{|I|}{|J|}}{\binom{M}{|J|}} \leq \frac{\binom{\lfloor \frac{2N}{N_v-1} \rfloor}{k}}{\binom{M}{k}}.$$

□

To minimize the number of multiplications done by the card, we use

case a. one prime number of length 32 bits to obtain a probability of cheating $\leq 1/2^{20}$,

case b. one prime number of length 75 bits to obtain a probability of cheating $\leq 1/2^{64}$.

Hence, to obtain with the help of the server the result of $a \times b \bmod c$ where a, b, c are 512 bits numbers and $a, b < c$, with a probability $> 1 - \frac{1}{2^{20}}$ in the case

a, and $> 1 - \frac{1}{2^{64}}$ in the case b the card

- computes the equivalent of around $\frac{1}{9.8}$ modular multiplication in the case a, and $\frac{1}{4}$ in the case b,
- exchanges 320 bytes with the server,
- needs 272 bytes of RAM in the case a, and 294 bytes in the case b,
- generates a random prime number with a size of 32 bits in the case a, and with a size of 75 bits in the case b.

Remark. If one wants to compute a modular multiplication and obtain a proof that the return value is correct with a probability $> 1 - \varrho$, one must change the prime number p used in the verification before each computation. Otherwise, the server could compute α^*, r^* satisfying $(a \times b) \bmod p_j = (\alpha^* \times c + r^*) \bmod p_j$ for a certain amount l_s of chosen p_j . Hence, if the card detects the error, for future computations, the server knows that the prime number chosen by the card does not belong to the chosen l_s . Then for the calculation of the probability, we must replace M by $M - l_s$ and so after some such rounds, the probability of cheating becomes $> \varrho$.

If the card uses a prime number of 75 bits then $|I| \leq 13$. Hence the server could not choose more than 13 prime numbers. Then after n verifications M could be replaced by $M - 13n$. Then if the number of verifications done by the card is less than 10^{18} the probability of cheating remains less than $\frac{1}{2^{64}}$. In this case the card could use a secret and unique prime number of 75 bits stored at the initialization.

If the card uses a prime number of 32 bits, the probability decreases much faster and the card must generate a new prime number before each computation. This can be done very efficiently using techniques developed by Couvreur and Quisquater [3].

3 Computation of $a^t \bmod c$

By repeated applications of the former protocol, the card is able to compute $a^t \bmod c$ in an efficient way. We suppose $a < c$.

We denote:

- $t = t_0 + t_1 2 + \dots + t_k 2^k = 2^{i_1} + \dots + 2^{i_\nu}$.
- $\text{VER}[a, b, \alpha, r, p_v]$ the verification: $(a \times b) \bmod p_v = (\alpha \times c + r) \bmod p_v$.

The protocol is the following one:

1. The card uses the randomly selected prime number p_v with $|p_v| = N_v$, computes and stores $c_p = c \bmod p_v$.
2. The card sends to the server a, t, c .
3. The server computes and sends to the card
 - $A = [\alpha_1, \dots, \alpha_k], R = [r_1, \dots, r_k]$
 - $B = [\beta_2, \dots, \beta_\nu], S = [s_2, \dots, s_\nu]$

such that

- for each $i \geq 1$

$$\alpha_i = \lfloor (r_{i-1} \times r_{i-1}) / c \rfloor,$$

$$r_i = (r_{i-1} \times r_{i-1}) \bmod c,$$

with $r_0 = a$,
- for each $j \geq 2$

$$\beta_j = \lfloor (r_{i_j} \times s_{j-1}) / c \rfloor,$$

$$s_j = (r_{i_j} \times s_{j-1}) \bmod c,$$

with $s_1 = r_{i_1}$.

4. The card verifies $\forall i \geq 1 \quad \text{VER}[r_{i-1}, r_{i-1}, \alpha_i, r_i, p_v]$,
 $\forall j \geq 2 \quad \text{VER}[r_{i_j}, s_{j-1}, \beta_j, s_j, p_v]$.
5. If there is no error during this last step, the card accepts: $a^t \bmod c := s_\nu$.

Theorem 3.1 *If a, c are non negative integers of length at most 512 bits such that $a < c$, then the probability of accepting a false value $a^t \bmod c$ is less than $1/2^{20}$ in the case a ($N_\nu = 32$) and less than $1/2^{64}$ in the case b ($N_\nu = 75$).*

Proof. Suppose $s_\nu \neq a^t \bmod c$. Hence, in the verification 4, there is at least one error. Then, according to theorem 2.1, the card detects this error with probability $> 1 - \frac{1}{2^{20}}$ in the case a, and $> 1 - \frac{1}{2^{64}}$ in the case b. □

Let us suppose that a and c have a size of 512 bits. During the step 4, the card computes:

- $r_{0,p} = a \bmod p_v$
for each $i \geq 1$, $d_{i,p} = r_{i-1,p} \times r_{i-1,p}$,
 $\alpha_{i,p} = \alpha_i \bmod p_v$,
 $e_{i,p} = \alpha_{i,p} \times c_p$,
 $r_{i,p} = r_i \bmod p_v$,
 $f_{i,p} = (r_{i,p} + e_{i,p} - d_{i,p}) \bmod p_v$,
the card verifies $f_{i,p} = 0$,
- $s_{1,p} = r_{i_1} \bmod p_v$
for each $j \geq 2$, $g_{j,p} = r_{i_j,p} \times s_{j-1,p}$,
 $\beta_{j,p} = \beta_j \bmod p_v$,
 $h_{j,p} = \beta_{j,p} \times c_p$,
 $s_{j,p} = s_j \bmod p_v$,
 $k_{j,p} = (s_{j,p} + h_{j,p} - g_{j,p}) \bmod p_v$,
the card verifies $k_{j,p} = 0$.

The number of operations done by the card is the following one:

- 2 modulo $512 \times N_\nu$ (computations of $c_p, r_{0,p}$),
- for each $i \in \{1, \dots, k\}$:
 - 2 modulo $512 \times N_\nu$ (computations of $\alpha_{i,p}, r_{i,p}$),
 - 2 multiplications $N_\nu \times N_\nu$ (computations of $d_{i,p}, e_{i,p}$),
 - 1 modulo $2N_\nu \times N_\nu$ (computation of $f_{i,p}$),
- for each $j \in \{2, \dots, \nu\}$:
 - 2 modulo $512 \times N_\nu$ (computations of $\beta_{j,p}, s_{j,p}$),
 - 2 multiplications $N_\nu \times N_\nu$ (computations of $g_{j,p}, h_{j,p}$),
 - 1 modulo $2N_\nu \times N_\nu$ (computation of $k_{j,p}$).

Then the card needs the equivalent of around

$$(2k + 2\nu) \frac{\theta}{2} \frac{512}{1024} \frac{N_\nu}{512} + (2k + 2\nu - 2) \frac{1}{2} \frac{N_\nu}{512} \frac{N_\nu}{512} + (k + \nu - 1) \frac{\theta}{2} \frac{2N_\nu}{1024} \frac{N_\nu}{512} \quad (4)$$

modular multiplications, that is, $\simeq \frac{k+\nu}{22}$ in the case a, and $\simeq \frac{k+\nu}{8}$ in the case b.

For the verification of vectors A, R , the card only needs memory for $r_{i-1,p}, \alpha_{i,p}, e_{i,p}, d_{i,p}$. Indeed $r_{i,p}$ uses the location for $r_{i-1,p}$, and $f_{i,p}$ is put instead of $\alpha_{i,p}$. The card stores $[r_{i_1,p}, \dots, r_{i_\nu,p}]$; similarly for the verification of vectors B, S the card needs some RAM only for $s_{j-1,p}, \beta_{j,p}, h_{j,p}, g_{j,p}$ stored at the location of $r_{i-1,p}, \alpha_{i,p}, e_{i,p}, d_{i,p}$.

The card stores $a, c, t, p_\nu, c_p, A, R, B, S, r_{i-1,p}, \alpha_{i,p}, e_{i,p}, d_{i,p}, r_{i_1,p}, \dots, r_{i_\nu,p}$; so it needs $(2k+2\nu) \times 64 + (8+\nu) \times \lceil \frac{N_\nu}{8} \rceil + \lceil \frac{k+1}{8} \rceil$ bytes of memory. It is possible to modify the protocol if the card has not enough memory. In this case, the server does not send the vectors A, R, B, S , but sends α_1, r_1 , and after the verification of the two values by the card, the server sends α_2, r_2 , *aso*. For this version of the protocol the card needs $4 \times 64 + (8+\nu) \times \lceil \frac{N_\nu}{8} \rceil + \lceil \frac{k+1}{8} \rceil$ bytes of memory.

Moreover there are $2(k+\nu) \times 64 + \lceil \frac{k+1}{8} \rceil$ exchanged bytes between the card and the server.

Thanks to the server, to obtain $a^t \bmod c$ where a, c are 512 bits numbers, with a probability $> 1 - \frac{1}{2^{20}}$ in the case a, and $> 1 - \frac{1}{2^{64}}$ in the case b, the card

- computes the equivalent of $\frac{k+\nu}{22}$ modular multiplications in the case a, and $\frac{k+\nu}{8}$ in the case b,
- exchanges $2(k+\nu) \times 64 + \lceil \frac{k+1}{8} \rceil$ bytes with the server,
- needs $256 + (8+\nu) \times \lceil \frac{N_\nu}{8} \rceil + \lceil \frac{k+1}{8} \rceil$ bytes of RAM,
- generates a random prime number with a size of 32 bits in the case a, and uses the secret prime number of 75 bits stored at the initialization in the case b.

These two protocols $a \times b \bmod c$ and $a^t \bmod c$ are very efficient to obtain the result with the corresponding proof of correctness; but all computations are public and are not convenient to produce a RSA signature or a signature using the discrete exponential. On the contrary, these protocols are very adequate to obtain fast verifications for RSA, DSS *aso*.

4 The discrete exponential

In this section, we show how the card can compute, thanks to a server, $a^x \bmod p$ with a a public and fixed integer, p a public and fixed prime number, and x secretly chosen by the card. Suppose that a, p are with a size of 512 bits such that $a < p$. Applications for this protocol are very important: signature of Schnorr [11], DSS signatures [7], *aso*. We use our previous protocol and the protocol of Brickell, Gordon, Mc Curley and Wilson [2].

The protocol BGCW

The goal of this protocol is to compute a^x for all $x \leq N$ using some precomputations.

If $x = \sum_{i=0}^{m-1} x_i b^i$ with $0 \leq x_i \leq h$, and if a^{b^i} is known for each i , then the algorithm for computing a^x is the following one:

```

 $B \leftarrow \prod_{x_i=h} a^{b^i}$ 
 $A \leftarrow B$ 
for  $d = h - 1$  to 1 by  $-1$ 
     $B \leftarrow B \times \prod_{x_i=d} a^{b^i}$ 
     $A \leftarrow A \times B$ 
return(A)

```

In our protocol, multiplications are modular multiplications, the $a_i = a^{b^i}$ are computed by the server, and A is computed by the card. To avoid the observation of the used time for a computation by an opponent which could give some information about x , it is necessary that the card uses a constant time for each computation: a solution is an algorithm with a constant number of multiplications, using, if necessary, the simulation (same time, no operation) of some multiplications.

During the computations, the card does it using one of the two methods:

case 1. The card stores $a^{b^0}, \dots, a^{b^{m-1}}$, next computes A . Let $k_d = \#\{x_i : x_i = d\}$; the number of multiplications done by the card during the algorithm is $(k_h - 1) + \sum_{d=1}^{h-1} (k_d + 1) = h - 2 + \sum_{d=1}^h k_d \leq h - 2 + m$. Some simulation of multiplications is needed; thus the card computes exactly the equivalent of $h - 2 + m$ multiplications.

case 2. Let $c_d = \prod_{x_i=d} a^{b^i}$. The card computes c_1, \dots, c_h while receiving the a^{b^i} 's. First $c_i = 1$ for each i , thus after receiving the first value there is no multiplication to do. But after this first step, in order to avoid to give any information to the server (or any opponent), if the received number has to be multiplied by 1, the card needs to simulate a complete multiplication. Thus the card needs exactly the equivalent of $m - 1$ multiplications to obtain c_1, \dots, c_h . Next the card needs $2(h - 1)$ multiplications to obtain A . The computations done by the card are exactly the equivalent of $2h - 3 + m$ multiplications.

Using $b_i = b^i$, we have $m = \lceil \log_b N \rceil$ and $h = b - 1$. The card computes a^x using:

case 1. $\lceil \log_b N \rceil + b - 3$ multiplications and storing $\lceil \log_b N \rceil - 1$ values $(a^{b^1}, \dots, a^{b^{m-1}})$,

case 2. $\lceil \log_b N \rceil + 2b - 5$ multiplications and storing $b - 1$ values (c_1, \dots, c_h) .

Our protocol is the following. Let x a number of the set $\{0, \dots, N\}$. For example, for the DSS, N is a prime number of length 160 bits. And suppose

$$x = \sum_{i=0}^{m-1} x_i b^i,$$

where $x_i \in \{0, \dots, b - 1\}$.

1. The card uses the randomly selected prime number p_v of length N_v bits, computes and stores $p_0 = p \bmod p_v$.
2. The server sends to the card, the numbers $a_i = a^{b^i} \bmod p$. We have $a_i = a_{i-1}^b \bmod p$ for all $i = 1, \dots, m-1$, with $a_0 = a$. Then the card knowing a can verify a_1 using the protocol of section 3, next it can verify a_2 aso. Then the card can calculate $a^x \bmod p$ with the protocol **BGCW**. It keeps in mind only the values it needs to compute $a^x \bmod p$.

Theorem 4.1 *If a, p are non negative integers of length at most 512 bits such that $a < p$, then the probability of accepting a false value $a^x \bmod p$ is less than $1/2^{20}$ in the case a and less than $1/2^{64}$ in the case b.*

Proof. Suppose $a^x \bmod p$ is not correct. Hence, at least one of the a_i is not correct. According to theorem 3.1, this would be detect with probability $> 1 - \frac{1}{2^{20}}$ in the case a or $> 1 - \frac{1}{2^{64}}$ in the case b. □

Performances

Let $k = |b| - 1$ and $\nu = \nu(b)$. Knowing $a_{i-1}, a_{i-1} \bmod p_v, p_0$, by (4) the card must do the equivalent of $\text{mult}(k, \nu) = (2k + 2\nu - 2) \frac{\theta}{2} \frac{512}{1024} \frac{N_v}{512} + (2k + 2\nu - 2) \frac{1}{2} \frac{N_v}{512} \frac{N_v}{512} + (k + \nu - 1) \frac{\theta}{2} \frac{2N_v}{1024} \frac{N_v}{512}$ modular multiplications to obtain and verify a_i for all $i = 1, \dots, m-1$. It must do $2 \frac{\theta}{2} \frac{512}{1024} \frac{N_v}{512}$ modular multiplications to obtain $a \bmod p_v$ and $p_0 = p \bmod p_v$. Hence it must do the equivalent of $(m-1) * \text{mult}(k, \nu) + \frac{\theta N_v}{1024}$ modular multiplications to obtain all the a_i and a proof that they are correct with a high probability. Hence to obtain $a^x \bmod p$ the card must do

- case 1.** $\lceil \log_b N \rceil + b - 3 + (\lceil \log_b N \rceil - 1) * \text{mult}(|b| - 1, \nu(b)) + \frac{\theta N_v}{1024}$ modular multiplications.
- case 2.** $\lceil \log_b N \rceil + 2b - 5 + (\lceil \log_b N \rceil - 1) * \text{mult}(|b| - 1, \nu(b)) + \frac{\theta N_v}{1024}$ modular multiplications.

Here, a and p are fixed, so they can be stored in the ROM. Then, the card must store for the two cases 130 bytes in the ROM (the numbers b, m, a, p). In the case b, the card must also store p_v (10 bytes).

The card stores in the EEPROM

- case 1.** $(\lceil \log_b N \rceil - 1) * 64 + \lceil \frac{|x|}{8} \rceil$ bytes : the numbers $a^b \bmod p, \dots, a^{b^{m-1}} \bmod p$ and x . It must write each number only one time for each signatures. Hence this is also the total number of bytes write in the EEPROM.
- case 2.** $(b-1) * 64 + \lceil \frac{|x|}{8} \rceil$ bytes : the numbers c_1, \dots, c_{b-1} and x . It must write each number c_i in average $\frac{m-1}{b-1}$ times for each signature and x only one time. Hence the total number of bytes write in the EEPROM is $(\lceil \log_b N \rceil - 1) * 64 + \lceil \frac{|x|}{8} \rceil$.

In the cases 1a and 2a the need RAM is $160 + 4 * \nu(b)$ bytes and in the cases 1b and 2b the need RAM is $208 + 10 * \nu(b)$

The exchange values between the card and the server are in the two cases $(\lceil \log_b N \rceil - 1) * 2 * (|b| + \nu(b) - 2) * 64$ bytes. Here we suppose that the server knows b, a, p , then the card does not send them to the server.

Security

The values given to the server are independent of x : then the protocol is secure against passive attacks.

If the server cheats to obtain some information, the card will see that with a probability $> 1 - \frac{1}{2^{20}}$ or $> 1 - \frac{1}{2^{64}}$. Then it will not use the value $a^x \text{ mod } p$. Hence the server does not obtain any information. Then the protocol is secure against active attacks.

In conclusion, our protocol is secure against passive and active attacks and the card obtains the good result with a probability $> 1 - \frac{1}{2^{20}}$ or $> 1 - \frac{1}{2^{64}}$.

Performances of our protocol for the DSS

Here N is a number of length 160 bits. Without the help of the server the card must do 240 modular multiplications on the average. We use in the first case $b = 16, m = 40$ and in the second $b = 8, m = 54$.

The following table gives for the two cases, the number of modular multiplications done by the card, the needed RAM, EEPROM and ROM (in bytes), the number of bytes exchanged between the card and the server and the factor of acceleration by using our protocol. We also give the total number of bytes write in the EEPROM during the protocol and the maximal average times we must write in each bytes.

	case 1a	case 2a	case 1b	case 2b
security	$1/2^{20}$	$1/2^{20}$	$1/2^{64}$	$1/2^{64}$
multiplications	60.1	72.3	67.8	85.2
RAM (bytes)	164	164	218	218
EEPROM (bytes)	2516	468	2516	468
Write (bytes)	2516	3392	2516	3392
Number of writing	1	7.6	1	7.6
ROM (bytes)	130	130	140	140
data transfers (bytes)	19968	20352	19968	20352
factor of acceleration	4	3.3	3.53	2.8

Let us notice that smart cards with fast data transfers (> 15 kbytes/s) are coming.

5 Conclusion

This paper shows how to use an insecure server for accelerating some useful cryptographic protocols. We present a protocol to obtain the modular exponentiation of the DSS: it's the first protocol who accelerate the discrete exponential modulo a prime number, secure against passive attacks without using precomputations. And it is also the first one secure against active attacks. Using a card of small memory, our protocol accelerates the DSS modular exponentiation with a factor 4 in the average without decreasing the security of the DSS scheme: this protocol gives no information about the secret value.

If the values a, b, p are standardized for a group of cards, the server only needs to store values: in fact the server may be an insecure external ROM. In particular, we could have a card with a second external ROM which does not need to be secure.

References

1. Anderson, R. J.: Attack on server-assisted authentication protocols. *Electronic Letters* (1992) p. 1473.
2. Brickell, E., Gordon, D. M., McCurley, K. S., Wilson, D.: Fast exponentiation with precomputation. In *Advances in Cryptology – Proceedings of Eurocrypt '92* (1993) vol. *Lecture Notes in Computer Science* 658 Springer-Verlag pp. 200–207.
3. Couvreur, C., Quisquater, J.-J.: An introduction to fast generation of large prime numbers. *Philips Journal of Research* (1982) pp. 231–264.
4. Kawamura, S., Shimbo, A.: Fast server-aided secret computation protocols for modular exponentiation. *IEEE Journal on selected areas communications* **11** (1993).
5. Matsumoto, T., Imai, H., Laih, C.-S., Yen, S.-M.: On verifiable implicit asking protocols for RSA computation. In *Advances in Cryptology – Proceedings of Auscrypt' 92* (1993) vol. *Lecture Notes in Computer Science* 718 Springer-Verlag pp. 296–307.
6. Matsumoto, T., Kato, K., Imai, H.: Speeding up secret computation with insecure auxiliary devices. In *Advances in Cryptology – Proceedings of Crypto '88* (1989) vol. *Lecture Notes in Computer Science* 403 Springer-Verlag pp. 497–506.
7. NIST: FIPS 186 for Digital Signature Standard (DSS).
8. Pfitzmann, B., Waidner, M.: Attacks on protocols for server-aided RSA computation. In *Advances in Cryptology – Proceedings of Eurocrypt '92* (1993) vol. *Lecture Notes in Computer Science* 658 Springer-Verlag pp. 153–162.
9. Quisquater, J.-J., Soete, M. D.: Speeding up smart card RSA computation with insecure coprocessors. In *Proceedings of Smart Cards 2000* (1989) pp. 191–197.
10. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* **21** (1978) pp. 120–126.
11. Schnorr, C.: Efficient identification and signatures for smart cards. In *Advances in Cryptology – Proceedings of CRYPTO '89* (1990) vol. *Lecture Notes in Computer Science* 435 Springer-Verlag pp. 235–251.
12. Yen, S.-M., Laih, C.-S.: More about the active attack on the server-aided secret computation protocol. *Electronic Letters* (1992) p. 2250.

This article was processed using the \LaTeX macro package with LLNCS style