Fast Server-Aided RSA Signatures Secure Against Active Attacks

Philippe Béguin[1,*] and Jean-Jacques Quisquater[2]

[1] Laboratoire d'Informatique **
Ecole Normale Supérieure
45 Rue d'Ulm, F–75 230 Paris Cédex 05, France
e-mail: Philippe.Beguin@ens.fr

[2] Laboratoire DICE
Université Catholique de Louvain
Place du Levant 3, B–1 348 Louvain-la-Neuve, Belgium
e-mail: Quisquater@dice.ucl.ac.be

Abstract. Small units like chip cards have the possibility of computing, storing and protecting data. Today such chip cards have limited computing power, then some cryptoprotocols are too slow. Some new chip cards with secure fast coprocessors are coming but are not very reliable at the moment and a little bit expensive for some applications. In banking applications there are few servers (ATM) relative to many small units: it is a better strategy to put the computing power into few large servers than into the not-very-often used cards.
A possible solution is to use the computing power of the (insecure) server to help the chip card. But it remains an open question whether it is possible to accelerate significantly RSA signatures using an insecure server with the possibility of active attacks: that is, when the server returns false values to get some part of secret from the card.
In this paper, we propose a new efficient protocol for accelerating RSA signatures, resistant against all known active and passive attacks. This protocol does not use expensive precomputations; the computation done by the card, the used RAM and the data transfers between the card and the server are small. With current chip cards it is thus possible to implement efficiently this protocol.

1 Introduction

Small devices, like chip cards or smart cards are easy to carry and have the possibility of computing, storing and protecting data. Unfortunately, today such

* 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.

D. Coppersmith (Ed.): Advances in Cryptology - CRYPTO '95, LNCS 963, pp. 57-69, 1995.

chip cards have limited computing power and some protocols are not executed in an efficient way: for example, public-key cryptographic protocols. Some new chip cards with fast and secure coprocessors are coming but are not reliable at the moment (due to problems of auxiliary memory) and are in some cases too expensive. Anyway, it is useful to have many cheap secure cards and to put the expensive part into one or few insecure servers.

A possible solution is to use an auxiliary unit such as 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. From a theoretical point of view it is interesting to study how to share the computing power of two parties with some security constraints. This paper is a new efficient contribution in this important field.

If the server is secure and will not leak the secrets, 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 then 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. Let us remark that many proposed protocols ([11],[16],[10]) did not have a strong verification step and then were broken ([2],[13],[9]).

There exist two kinds of attacks against such protocols: classical searching ones are called *passive attacks*; specific ones where the server returns false values to get some information from the card, are called *active attacks*.

Such protocols were first studied by Matsumoto, Kato and Imai [11] and by Quisquater and De Soete [14] for accelerating RSA signatures [15]. Next Pfitzmann and Waidner [13] proposed some passive attacks against all protocols presented in [11]. And Anderson [2] proposed a very efficient active attack against one of the two protocols presented in [11], where a dishonest server could obtain the secret key, by using only one false signature. But these attacks could be very easily defeated if the card checks the correctness of the computed signatures and by increasing the parameters used in [11]. On the other hand Quisquater and De Soete's protocol is provably secure against passive attacks but it is very less efficient than Matsumoto, Kato and Imai's ones. Then Yen and Laih [16] and Matsumoto, Imai, Laih and Yen [10] presented an improvement of protocols presented in [11] secure against passive attacks: unfortunately they need expensive precomputations and thus, they are not very efficient. Next Kawamura and Shimbo [6] proposed four protocols provably secure against passive attacks: the two first are not very efficient and the two last are more efficient but they need expensive precomputations. So, all previously known protocols secure against passive attacks are not efficient or need expensive precomputations. Moreover, *absolutely none* of these protocols (provably or not provably secure against passive attacks) are secure against active attacks presented in [13]. As it is said in [6], it was an open question whether it is possible or not to construct a secure protocol against active attacks.

Burns and Mitchell [5] construct improvements of the two protocols presented in [11] (RSA-S1 and RSA-S2) secure against active attacks. Unfortunately, the first one (RSA-S1) is inefficient: the card has to do at least 188 modular multiplications for each signature. Otherwise, the second one (RSA-S2) is much efficient but it ignores Pfitzmann and Waidner's attacks [13] and thus is not secure.

Lim and Lee [9] developed other protocols using precomputations, based on the two-phase protocols due to Matsumoto, Imai, Laih and Yen [10].

Otherwise, Béguin and Quisquater [3] give the first method for accelerating significantly DSS signatures [12] provably secure against both passive and active attacks.

But it remains an open question whether it is possible to accelerate significantly (without expensive precomputations) RSA signatures using an insecure server, in a secure way against both passive and active attacks. In this paper, we propose a new efficient protocol resistant against all known passive and active attacks, including those presented in [13]. Moreover, we will show that our protocol is secure against more specific passive attacks. This protocol does not use expensive precomputations; the computation done by the card, the used RAM and the data transfers between the card and the server are small. Then, it is possible to implement efficiently this protocol with current chip cards.

We begin by giving some preliminaries, then we outline the protocol of Brickell, Gordon, McCurley and Wilson [4], which we will use. Then we describe our protocol and study its security. Finally, we expose the performances of this protocol.

2 Preliminaries

We denote by n the public modulus of RSA ($n = p \cdot q$), by s the secret signature key and by v the public verification key such that $s \cdot v = 1 \bmod \phi(n)$ with $\phi(n) = (p-1)(q-1)$. The card receives the message M and wants to compute, using the server, the signature of M: i.e. $S = M^s \bmod n$.

For a number a, we denote by $l(a)$ the number of bits of a, i.e. $l(a) = \lfloor \log_2 a \rfloor + 1$, and for a set F, we denote by $\#F$ the cardinality of F.

Let $k = l(n)$ and let $t = \max(l(p), l(q)) - 1$. In this paper, we will study the acceleration of the RSA signatures with 512 bit numbers ($k = 512$) and with 768 bit numbers ($k = 768$). We denote by modular multiplication the multiplication of two k bit numbers modulo a k bit number. In this paper, the computations done by the card will be measured in terms of modular multiplications. Hence we consider that the computation of $a \times b$ is about the equivalent of $\frac{1}{2} \times \frac{l(a)}{k} \times \frac{l(b)}{k}$ modular multiplications and the computation of $a \bmod b$ is about the equivalent of $\frac{\theta}{2} \times \frac{l(a)}{2k} \times \frac{l(b)}{k}$ modular multiplications. By θ we denote a constant near 1 varying with the implementation of the operation modulo. Here we use the practical value of 1.25 for θ. We suppose that the implementation of multiplications

modulo is classical and does not use clever tricks or the Montgomery's method: such methods need a specific study.

3 Fast Exponentiation with Precomputation

Several protocols are known to compute exponentiation with precomputation [4], [8]... We present a protocol due to Brickell, Gordon, McCurley and Wilson [4]. The goal of this protocol is to compute a^x using some precomputations.

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

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

In our protocol, the x_i will be known by the server which computes a^{x_i}, but x must be kept secret. Then, the card must use constant time to obtain a^x; otherwise, by observing the time used to compute a^x, an opponent could obtain some information about x. 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 uses one of the two following methods.

Case 1. The card stores $a^{x_0}, \ldots, a^{x_{m-1}}$, next computes A. Let $k_d = \#\{a_i : a_i = d\}$; 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 $c_d = \prod_{a_i=d} a^{x_i}$. The card computes $c_1, \ldots, c_h$ while receiving the a^{x_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 to 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 $c_1, \ldots, c_h$. Next the card needs $2(h-1)$ multiplications to obtain A. Hence the card computes exactly the equivalent of $2h - 3 + m$ multiplications.

4 The RSA Signature

Using the Extended Euclidean Algorithm, the values w_p, w_q can be computed such that $w_p + w_q = 1$, $w_p \bmod p = 0$, $w_q \bmod q = 0$ and $0 \leq |w_p|, |w_q| \leq n$. Thus, if $y_p \equiv y \bmod p$ and $y_q \equiv y \bmod q$, we have $y \equiv y_p w_q + y_q w_p \bmod n$. The protocol is the following.

1. The card receives M to sign.
2. The card chooses randomly $a_0, a_1, \ldots, a_{m-1}$ s. t. $a_i \in \{0, \ldots, h\}$,
   $x_0, \ldots, x_{m-1}$ s. t. $l(x_i) \leq t - \log_2(m \cdot h) - 2$.
3. The card computes $s_1 = \sum_{i=0}^{m-1} a_i x_i$.
4. The card sends M, n and $x_0, \ldots, x_{m-1}$ to the server.
5. The server returns $z_0, \ldots, z_{m-1}$ such that $z_i = M^{x_i} \bmod n$.
6. The card computes $z_p = \prod_{i=0}^{m-1} z_i^{a_i} \bmod p$ and $z_q = \prod_{i=0}^{m-1} z_i^{a_i} \bmod q$ using the algorithm [BGMW] already described.
7. The card computes $s_2 = s - s_1$,
   represents s_2 under the form $\sigma_p = s_2 \bmod (p-1) + \varrho_p(p-1)$,
   $\sigma_q = s_2 \bmod (q-1) + \varrho_q(q-1)$,
   where ϱ_p is a random number of $\{0, \ldots, q-2\}$,
   ϱ_q is a random number of $\{0, \ldots, p-2\}$.
8. The card sends to the server σ_p, σ_q.
9. The server computes and sends to the card $y_p = M^{\sigma_p} \bmod n$
   $y_q = M^{\sigma_q} \bmod n$.
10. The card computes $S_p = y_p \times z_p \bmod p$ and $S_q = y_q \times z_q \bmod q$.
11. Now the card computes $S = w_q \cdot S_p + w_p \cdot S_q \bmod n$.
12. The card verifies $S^v \bmod n = M$.
13. If during the step 12, the verification is correct, then the card transmits S.

5 Security

5.1 An Exhaustive Search

A possible attack is to make an exhaustive search over s_1. An attacking server knowing M, S, y_p, computes $\gcd(n, S - y_p M^{\sum_{i=0}^{m-1} a_i x_i} \bmod n)$ for all a_i such that $0 \leq a_i \leq h$.
When $\sum_{i=0}^{m-1} a_i x_i = s_1$, this server obtains $\gcd(n, S - y_p M^{\sum_{i=0}^{m-1} a_i x_i} \bmod n) = p$ with very high probability. The complexity of this attack is $(h+1)^m$.

We have $\sigma_p = (s - s_1) \bmod (p-1) + \varrho_p(p-1)$. Hence $v\sigma_p = (1 - vs_1) \bmod (p-1) + v\varrho_p(p-1)$. But $(p-1) \bmod 2 = 0$, then $\sum_{i=0}^{m-1} (vx_i \bmod 2)(a_i \bmod 2) = 1 + v\sigma_p \bmod 2$ is known. Hence, using this equation, the complexity of the exhaustive search becomes $(h+1)^m/2$.

We suppose that the other factors of $p-1$ and $q-1$ are unknown (which is a normal supposition for RSA signatures), hence it is not possible to improve this search.

5.2 A Pseudo-Exhaustive Search

This attack is similar to the Pfitzmann and Waidner's attack [13] against the protocols of Matsumoto, Kato and Imai [11].
For all $A = \{a_0, \ldots, a_{\lceil m/2 \rceil - 1}\}$ such that $0 \leq a_i \leq h$, the attacking server computes

$$y_A = M^{\sum_{i=0}^{\lceil m/2 \rceil - 1} a_i x_i} \bmod n.$$

For all $B = \{b_{\lceil m/2 \rceil}, \ldots, b_{m-1}\}$ such that $0 \leq b_i \leq h$, the server computes

$$z_B^* = S \cdot y_p^{-1} \cdot z_B^{-1} \bmod n \qquad \text{with} \qquad z_B = M^{\sum_{i=\lceil m/2 \rceil}^{m-1} b_i x_i} \bmod n.$$

We have

$$s_1 = \sum_{i=0}^{m-1} a_i x_i = \sum_{i=0}^{\lceil m/2 \rceil - 1} a_i x_i + \sum_{j=\lceil m/2 \rceil}^{m-1} a_j x_j,$$

with $0 \leq a_i, a_j \leq h$. Let $A = \{a_0, \ldots, a_{\lceil m/2 \rceil - 1}\}$ and $B = \{a_{\lceil m/2 \rceil}, \ldots, a_{m-1}\}$, then $M^{s_1} \bmod n = y_A \cdot z_B \bmod n$. Thus

$$z_B^* \bmod p = \left(S \cdot y_p^{-1} \cdot z_B^{-1} \bmod n\right) \bmod p = S \cdot y_p^{-1} \cdot M^{-s_1} \cdot y_A \bmod p = y_A \bmod p.$$

Hence, with very high probability $\gcd(y_A - z_B^*, n) = p$.

Let $X = (h+1)^{\lceil m/2 \rceil}$, we denote by $y_1, \ldots, y_X$ and $z_1^*, \ldots, z_X^*$ respectively all the possible values of y_A and z_B^*.
We here describe two ways to perform this attack.

Way a. For all i and j, the attacking server computes $\gcd(y_i - z_j^*, n)$. The complexity of this attack is

$$X^2 = \left((h+1)^{\lceil m/2 \rceil}\right)^2.$$

Way b. Define the polynomial

$$P(z) = \prod_{j=1}^{X} \left(z - z_j^*\right) \bmod n.$$

For all $i \in \{1, \ldots, X\}$, the attacking server computes $P(y_i) \bmod n$. Using FFT algorithms described in [1] (chapter 8), we can theoretically evaluate the polynomial P in X points in $X(\log X)^2$ steps. Hence the complexity of this attack becomes

$$(h+1)^{\lceil m/2 \rceil} \left(\log\left((h+1)^{\lceil m/2 \rceil}\right)\right)^2.$$

The second way needs a lot of cache and RAM memory. Theoretically this way b's attack needs $X(\log X)^2$ steps, but practically the needed hundred Gigabytes will be stored on "slow" discs instead of fast access-memory. So, the practical complexity of the second way is quite the same as the complexity of the first one. Hence, the parameters used in section 6.2 to counter way b's attack will be more secure than we need.

5.3 An Attack Using the LLL Algorithm [7]

We consider that the card computes several signatures. We denote by $s_1^{[i]}$ the value s_1 used during the i^{th} signature. Let us suppose that $l(s_1^{[i]}) \leq \frac{t}{\alpha} - 2$ for all i with $\alpha > 1$. We suppose without lost of generality $l(p) = t + 1$.
We have $\sigma_p^{[i]} = s - s_1^{[i]} + \varrho_p^{[i]}(p-1)$, then $\sigma_p^{[i]} - \sigma_p^{[i+1]} = s_1^{[i+1]} - s_1^{[i]} + (\varrho_p^{[i]} - \varrho_p^{[i+1]})(p-1)$. Let us use the following notations $\sigma_i = \sigma_p^{[i]} - \sigma_p^{[i+1]}$, $r_i = s_1^{[i+1]} - s_1^{[i]}$ and $\varrho_i = \varrho_p^{[i]} - \varrho_p^{[i+1]}$; we have $\sigma_i = r_i + \varrho_i(p-1)$. The server knows σ_i for all i, but r_i, ϱ_i and $p-1$ are unknown. We have $l(\sigma_i) = k$, $|\varrho_i| \leq q-1$, and $|r_i| \leq 2^{\frac{t}{\alpha}-2}$. Consider the following matrix where l is a non negative integer:

$$A = \begin{pmatrix} \sigma_2 & \sigma_3 & \cdots & \sigma_{l+1} & \varepsilon \\ \sigma_1 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \sigma_1 & 0 \end{pmatrix}$$

where ε is a random number of size $\leq \frac{t}{\alpha} - 1$.
Consider the left multiplication by the vector $Y_0 = (\varrho_1, -\varrho_2, \ldots, -\varrho_{l+1})$, we obtain the vector $(\varrho_1\sigma_2 - \varrho_2\sigma_1, \varrho_1\sigma_3 - \varrho_3\sigma_1, \ldots, \varrho_1\sigma_{l+1} - \varrho_{l+1}\sigma_1, \varrho_1\varepsilon)$.
But $\varrho_1\sigma_i - \varrho_i\sigma_1 = \varrho_1 r_i - \varrho_i r_1$, then $|\varrho_1\sigma_i - \varrho_i\sigma_1| \leq q2^{\frac{t}{\alpha}-1}$ and $|\varrho_1\varepsilon| \leq q2^{\frac{t}{\alpha}-1}$.
Let $Y = (y_1, -y_2, \ldots, -y_{l+1})$ be a non null vector. Then

$$(y_1, -y_2 \ldots, -y_{l+1})\, A = (y_1\sigma_2 - y_2\sigma_1, \ldots, y_1\sigma_{l+1} - y_{l+1}\sigma_1, y_1\varepsilon)\,.$$

If y_1 is much larger than q then the last element will have a too large size. So consider all values of y_1 smaller than $4q$.
We have $y_1\sigma_i = \lfloor \frac{y_1\sigma_i}{\sigma_1} \rfloor \sigma_1 + r$, then $y_1\sigma_i - y_i\sigma_1 = \left(\lfloor \frac{y_1\sigma_i}{\sigma_1} \rfloor - y_i \right) \sigma_1 + r$. The LLL algorithm will minimize $y_1\sigma_i - y_i\sigma_1$ for all i, then $y_i = \lceil \frac{y_1\sigma_i}{\sigma_1} \rfloor$. Hence the i^{th} coordinate of $Y \times A$ will be $\sigma_i y_1 \bmod \sigma_1$ if $\sigma_i y_1 \bmod \sigma_1 \leq \frac{\sigma_1}{2}$ and $-(\sigma_1 - \sigma_i y_1 \bmod \sigma_1)$ if $\sigma_i y_1 \bmod \sigma_1 \geq \frac{\sigma_1}{2}$.
Consider the following probability:

$$\begin{aligned} \Pr_{y_1 < 4q} \left(\sigma_i y_1 \bmod \sigma_1 < q2^{\frac{t}{\alpha}-1} \text{ or } (\sigma_1 - \sigma_i y_1) \bmod \sigma_1 < q2^{\frac{t}{\alpha}-1} \right) &\simeq 2\frac{q2^{\frac{t}{\alpha}-1}}{\sigma_1} \\ &\simeq \frac{q2^{\frac{t}{\alpha}}}{pq} \\ &= \frac{2^{\frac{t}{\alpha}}}{p}, \end{aligned}$$

then

$$\Pr_{y_1 < 4q} \left(\sigma_i y_1 \bmod \sigma_1 < q2^{\frac{t}{\alpha}} \text{ or } (\sigma_1 - \sigma_i y_1) \bmod \sigma_1 < q2^{\frac{t}{\alpha}} \quad \forall i \right) \simeq \left(2^{\frac{t}{\alpha}}/p \right)^l.$$

Then the average number of integers $y_1 < 4q$ such that $\sigma_i y_1 \bmod \sigma_1 < q2^{\frac{t}{\alpha}}$ or $(\sigma_1 - \sigma_i y_1) \bmod \sigma_1 < q2^{\frac{t}{\alpha}}$ for all i is about $4q\left(2^{\frac{t}{\alpha}}/p\right)^l$.

We have $l(p) = t + 1$ then $p \geq 2^t$. Hence, if $\alpha > 1$, there exists l such that $4q\left(2^{\frac{t}{\alpha}}/p\right)^l \ll 1$. We have proven that there exists a vector Y_0 such that $y_1\sigma_i - y_i\sigma_1 \leq q2^{\frac{t}{\alpha}}$ for all i: the vector $Y_0 = (\varrho_1, -\varrho_2, \ldots, -\varrho_{l+1})$. The probability that there exists another such a vector Y is very small; then the LLL algorithm will find the vector Y_0.

But in our protocol, we have $s_1 = \sum_{i=0}^{m-1} a_i x_i$, with $l(x_i) \leq t - \log_2(mh) - 2$ and $0 \leq a_i \leq h$, then $0 \leq s_1 \leq mh2^{t-\log_2(mh)-2}$. Hence $l(s_1) \leq t - 2$. Then, in our protocol $\alpha = 1$, and we cannot find such an l, hence this attack is ineffective.

5.4 Modification of the Previous Attack

In our protocol $l(s_1^{[i]}) \leq t - 2$ for all i; then $l(s_1^{[i]}) \leq t - \beta - 2$, with probability $1/2^\beta$. Then to find an $s_1^{[i]}$ such that $l(s_1^{[i]}) \leq t - \beta - 2$, we must consider 2^β different $\sigma_p^{[i]}$. But in the previous attack, we must obtain $l+1$ different $\sigma_p^{[i]}$ such that $l(s_1^{[i]}) \leq t/\alpha - 2$; here $\alpha = \frac{t}{t-\beta}$. Hence we must obtain $(l+1)2^\beta$ different $\sigma_p^{[i]}$; next, we must find the $l+1$ values $\sigma_p^{[i]}$ such that $l(s_1^{[i]}) \leq t/\alpha - 2$: there are $\binom{(l+1)2^\beta}{l+1}$ possibilities, and for each we must perform a LLL reduction. Then the total number of LLL reductions done is $\binom{(l+1)2^\beta}{l+1}$.

Consider for simplicity $l(p) = l(q) = t + 1 = k/2$. Then the constraint $q\left(2^{\frac{t}{\alpha}}/p\right)^l \ll 1$ will become $t + tl(1/\alpha - 1) \ll 0$, thus $1 + l(1/\alpha - 1) < 0$. Hence $l > \alpha/(\alpha - 1) = t/\beta$. Then, the workload w of this attack satisfies

$$w > w(\beta) = \binom{\frac{t}{\beta}2^\beta}{\frac{t}{\beta}}.$$

With RSA-512 ($k = 512$) the minimum is obtained when $\beta = 129$: $w(129) = 2^{130}$. And with RSA-768 ($k = 768$) the minimum is obtained when $\beta = 193$: $w(193) = 2^{194}$. Recall that we must perform $w(\beta)$ LLL reductions, then these attacks are ineffective.

5.5 Active Attacks

If the server cheats to obtain some information, the card will detect that. Then the card will not reveal the false value of S. Hence an active attack using only one false signature like the Anderson's one [2] is impossible.

Moreover, since s_1 and the x_i's are different for each signature, active attacks in several rounds like Pfitzmann and Waidner's ones [13] are impossible. The main difference between our protocol and all previously known ones ([11], [14], [10], [6], [16]...) is the use of random values which prevents our protocol from active attacks using several signatures.

6 Performances

We suppose $v = 3$. We also suppose for evaluating the computations done by the card that $l(p) = k/2$ and $l(q) = k/2$. We use the results explained in section 2; all values used here are clearly more precise than needed, but in order to overcome the addition of error factors, we will approximate the results only at the end of the analysis.

6.1 Computations done by the Card

The card must multiply a $k/2$ bit number (say y) by a k bit number (say z) modulo a $k/2$ bit number (say p). The best way is to compute $z \bmod p$, then multiply the two $k/2$ bit numbers and then take the result modulo p. With this way the computation is $\frac{\theta}{2} \cdot \frac{k}{2k} \cdot \frac{k/2}{k} + \frac{1}{2} \cdot \frac{k/2}{k} \cdot \frac{k/2}{k} + \frac{\theta}{2} \cdot \frac{k}{2k} \cdot \frac{k/2}{k} = \frac{7}{16}$ modular multiplication.
The card must also multiply two $k/2$ bit numbers modulo a $k/2$ bit number which takes $\frac{1}{2} \cdot \frac{k/2}{k} \cdot \frac{k/2}{k} + \frac{\theta}{2} \cdot \frac{k}{2k} \cdot \frac{k/2}{k} = \frac{9}{32}$ modular multiplication.

To obtain z_p and z_q:

case 1 The card must do two modulo $(k, k/2)$ to obtain $z_0 \bmod p$ and $z_0 \bmod q$, then $m - 1$ multiplications $(k/2, k)$ modulo p and modulo q, and $h - 1$ multiplications $(k/2, k/2)$ modulo p and modulo q to obtain z_p and z_q. Hence the card must do $\frac{5}{16} + (m-1) \times \frac{7}{8} + (h-1) \times \frac{9}{16}$ modular multiplications.

case 2 The card must do two modulo $(k, k/2)$ to obtain $z_0 \bmod p$ and $z_0 \bmod q$, then $m-1$ multiplications $(k/2, k)$ modulo p and modulo q to obtain $c_1, \ldots c_h$ modulo p and q, and $2h-2$ multiplications $(k/2, k/2)$ modulo p and modulo q to obtain z_p and z_q. Hence the card must do $\frac{5}{16} + (m-1) \times \frac{7}{8} + (2h-2) \times \frac{9}{16}$ modular multiplications.

The card must do $\frac{1}{4}$ modular multiplication to obtain σ_p and σ_q: we suppose that $s \bmod (p-1)$ and $s \bmod (q-1)$ are stored in memory; next it must do $2 \times \frac{7}{16}$ modular multiplication to obtain S_p and S_q, and $2 \cdot \frac{1}{2} \cdot \frac{k/2}{k} \cdot \frac{k}{k} + \frac{\theta}{2} \cdot \frac{3k/2}{2k} \cdot \frac{k}{k} = \frac{31}{32}$ to obtain S. Finally the card needs two modular multiplications to verify S.

Hence the total number of modular multiplications done by the card is

case 1 $\frac{5}{16} + (m-1) \times \frac{7}{8} + (h-1) \times \frac{9}{16} + \frac{67}{32} + 2,$

case 2 $\frac{5}{16} + (m-1) \times \frac{7}{8} + (2h-2) \times \frac{9}{16} + \frac{67}{32} + 2.$

6.2 Choice of the Parameters

In order to leave the attacks studied in section 5 ineffective, that is of complexity at most 2^{64}, and to minimize the number of modular multiplications done by the card, we take

way a
case 1 $h = 10, m = 19,$
case 2 $h = 7, m = 22.$
way b
case 1 $h = 17, m = 25,$
case 2 $h = 11, m = 29.$

6.3 The Needed Memory of the Card

We will evaluate the amount of memory the card needs to achieve the above protocol. We separate this memory into RAM and EEPROM.

In EEPROM are put all fixed values $s, s \bmod (p-1), s \bmod (q-1), v, n, p, q, w_p$, w_q, that is $6k+1$ bytes, and some other values.

case 1 M and $z_0, \ldots, z_{m-1}$ are stored in the EEPROM that is $(m+1) \times k$ bits. The total number of bits written into the EEPROM for each signature is $(m+1) \times k$, and each bit of memory is used only one time for each signature.
case 2 M and $c_{1,p}, \ldots, c_{h,p}, c_{1,q}, \ldots, c_{h,q}$ are stored in the EEPROM, that is $(h+1) \times k$ bits. When the card receives a new z_i, it must compute the new corresponding $c_{i,p}$ and $c_{i,q}$ and write these new values into the EEPROM. Then the total number of bits written into the EEPROM is $(m+1) \times k$, and each bit of memory is used at most $\frac{m}{h}$ times for each signature.

Hence the need EEPROM is

case 1 $(m+7) \times k + 1$ bits,
case 2 $(h+7) \times k + 1$ bits.

We consider now the RAM. In step 2., the card chooses a_0 and x_0 and computes $s_1 = a_0 x_0$, then sends x_0 to the server. Then it chooses a_1, x_1 and computes $s_1 := s_1 + a_1 x_1$ ... The card keeps in mind $s_1, a_0, \ldots, a_{m-1}, x_i$. Hence it is easy to see that the maximum needed RAM for the card is in step 6. In step 6., the card must store

case 1 $s_1, a_0, \ldots, a_{m-1}, A, B, z_i, n$: the needed RAM is $4k + t - 2 + m \log_2(h+1)$ bits.
case 2 $s_1, a_0, \ldots, a_{m-1}, A_p, B_p, A_q, B_q, z_i, p, q$. To obtain the new A_p, B_p, the card must compute and store $z_i \bmod p$, next to obtain the new A_q, B_q, it must compute and store $z_i \bmod q$ which can be put instead of $z_i \bmod p$. Then the needed RAM is $t - 2 + m \log_2(h+1) + 4k + t + 1$ bits.

6.4 The Data Transfers

In the two cases, the card must send to the server $M, n, x_0, \ldots, x_{m-1}, \sigma_p, \sigma_q$, and the server must send to the card $z_0, \ldots, z_{m-1}, y_p, y_q$. Then the data transfers are $(6+m) \times k + m \times (t - \log_2(mh) - 2)$ bits.

6.5 Computations done by the Server

All computations done by the server are the evaluations of $M^x \bmod n$ for several $x \leq N$. It will do it by using the protocol BGMW [4] already describe with $x_i = b^i$, $m = \lceil \log_b N \rceil$ and $h = b - 1$ for an appropriate b. Then knowing $M^{b^i} \bmod n$ for all $i = 0, \ldots, m-1$, the number of modular multiplications done by the server is in the average $\frac{b-1}{b}\lceil \log_b N \rceil + b - 3$. In our protocol, the server must do this for $N = 2^{\frac{k}{2}}$ (step 5.) and for $N = 2^k$ (step 9.).

RSA 512: $k = 512$
The server computes and stores $M^{16^i} \bmod n$ for $i = 0, \ldots, 63$ and $M^{32^i} \bmod n$ for $i = 0, \ldots, 102$ which needs 510 modular squares. Then in the step 5. for each modular exponentiation, the server must do in the average 73 modular multiplications using $b = 16$, and in the step 9. for each modular exponentiation, in the average 128.8 modular multiplications using $b = 32$. Then the total number of modular multiplications done by the server is $510 + 73 \times m + 2 \times 128.8$.

RSA 768: $k = 768$
The server computes and stores $M^{16^i} \bmod n$ for $i = 0, \ldots, 95$ and $M^{32^i} \bmod n$ for $i = 0, \ldots, 153$ which needs 765 modular squares. Then in the step 5. for each modular exponentiation, the server must do in the average 103 modular multiplications using $b = 16$, and in the step 9. for each modular exponentiation, in the average 178.2 modular multiplications using $b = 32$. Then the total number of modular multiplications done by the server is $765 + 103 \times m + 2 \times 178.2$.

6.6 Results

The following tables give for the two cases, the number of modular multiplications done by the card and by the server, the needed RAM, EEPROM (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 written into the EEPROM during the protocol and the maximal average times we must write in each bytes. We take for t the value $k/2 - 1$.

In the first table, we give results for the RSA 512 bits, and in the second for the RSA 768 bits. We recall that, using the Chinese Remainder Theorem, the card can compute a RSA signature in 260 modular multiplications when $k = 512$ and in 388 when $k = 768$.

Let us notice that today it is possible to write into the EEPROM in parallel with other computations (without any penalty of time) and to use a data transfer of 100 kbits/s.

	way a		way b	
	case 1	case 2	case 1	case 2
multiplications (card)	25	30	35	40
without server	260			
factor of acceleration	10.4	8.7	7.4	6.5
EEPROM (bytes)	1665	897	2049	1153
write (bytes)	1280	1472	1664	1920
number of writing	1	3.1	1	2.6
RAM	296	328	301	333
data transfers (bytes)	2183	2468	2748	3127
multiplications (server)	2155	2374	2593	2885

Table 1. RSA–512

	way a		way b	
	case 1	case 2	case 1	case 2
multiplications (card)	25	30	35	40
without server	388			
factor of acceleration	15.5	12.9	11.1	9.7
EEPROM (bytes)	2497	1345	3073	1729
write (bytes)	1920	2208	2496	2880
number of writing	1	3.1	1	2.6
RAM	440	488	445	493
data transfers (bytes)	3287	3716	4140	4711
multiplications (server)	3078	3387	3696	4108

Table 2. RSA–768

7 Conclusion

We have presented a new efficient protocol for accelerating RSA signatures using an insecure and fast server. This protocol is resistant against all known active and passive attacks. It does not use expensive precomputations; the computation done by the card, the needed RAM and the data transfers between the card and the server are small. Then, it is possible to implement efficiently this protocol with current chip cards.

It remains an open question: the existence of efficient protocols (without the use of precomputation) for accelerating RSA signatures *provably* secure against passive and active attacks.

References

1. Aho, A. V., Hopcroft, J. E., Ullman, J. D.: The design and analysis of computer algorithms. Addison-Wesley, Reading, Mass. (1974).
2. Anderson, R. J.: Attack on server-assisted authentication protocols. Electronic Letters (1992) p. 1473.
3. Béguin, P., Quisquater, J.-J.: Secure acceleration of DSS signatures using insecure server. In Proceedings of Asiacrypt '94 (To appear).
4. Brickell, E., Gordon, D. M., McCurley, K. S., Wilson, D.: Fast exponentiation with precomputation. In Advances in Cryptology – Proceedings of Eurocrypt '92 (1993) Lecture Notes in Computer Science vol. 658 Springer-Verlag pp. 200–207.
5. Burns, J., Mitchell, C. J.: Parameter selection for server-aided RSA computation schemes. IEEE Transactions on computers **43** (1994) pp. 163–174.
6. Kawamura, S., Shimbo, A.: Fast server-aided secret computation protocols for modular exponentiation. IEEE Journal on selected areas communications **11** (1993).
7. Lenstra, A. K., Lenstra, H. W., Lovász, L.: Factoring polynomials with rational coefficients. Math. Ann. **261** (1982) pp. 515–534.
8. Lim, C. H., Lee, P. J.: More flexible exponentiation with precomputation. In Advances in Cryptology – Proceedings of Crypto '94 (1994) vol. Lecture Notes in Computer Science 839 Springer-Verlag pp. 95–107.
9. Lim, C. H., Lee, P. J.: Security and performance of server-aided rsa computation protocols. In this Proceedings .
10. 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) Lecture Notes in Computer Science vol. 718 Springer-Verlag pp. 296–307.
11. Matsumoto, T., Kato, K., Imai, H.: Speeding up secret computation with insecure auxiliary devices. In Advances in Cryptology – Proceedings of Crypto '88 (1989) Lecture Notes in Computer Science vol. 403 Springer-Verlag pp. 497–506.
12. NIST: FIPS 186 for Digital Signature Standard (DSS).
13. Pfitzmann, B., Waidner, M.: Attacks on protocols for server-aided RSA computation. In Advances in Cryptology – Proceedings of Eurocrypt '92 (1993) Lecture Notes in Computer Science vol. 658 Springer-Verlag pp. 153–162.
14. Quisquater, J.-J., De Soete, M.: Speeding up smart card RSA computation with insecure coprocessors. In Proceedings of Smart Cards 2000 (1989) pp. 191–197.
15. 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.
16. Yen, S.-M., Laih, C.-S.: More about the active attack on the server-aided secret computation protocol. Electronic Letters (1992) p. 2250.