Security and Performance of Server-Aided RSA Computation Protocols

Chae Hoon Lim and Pil Joong Lee

Department of Electrical Engineering
Pohang University of Science and Technology (POSTECH)
Pohang, 790-784, KOREA
E-mail : lch@baekdu.postech.ac.kr ; pjl@vision.postech.ac.kr

Abstract. This paper investigates various security issues and provides possible improvements on server-aided RSA computation schemes, mainly focused on the two-phase protocols, RSA-S1M and RSA-S2M, proposed by Matsumoto et al. [4]. We first present new active attacks on these protocols when the final result is not checked. A server-aided protocol is then proposed in which the client can check the computed signature in at most six multiplications irrespective of the size of the public exponent. Next we consider multi-round active attacks on the protocol with correctness check and show that parameter restrictions cannot defeat such attacks. We thus assume that the secret exponent is newly decomposed in each run of the protocol and discuss some means of speeding up this preprocessing step. Finally, considering the implementation-dependent attack, we propose a new method for decomposing the secret and performing the required computation efficiently.

1 Introduction

Smart cards in popular use throughout the world do not have any dedicated crypto-engine and thus are not powerful enough to perform complicated computations, such as modular exponentiation, required for most public key cryptosystems (e.g., signature generation with RSA and signature verification in ElGamal-like signature schemes). To speed up such computations by a weak power smart card, much research has been conducted under the subject called the server-aided secret computation (SASC) [1-12]. In the SASC protocol, the client (the smart card) wants to perform a secret computation (e.g., RSA signature generation) by borrowing the computing power of an untrusted powerful server without revealing its secret information.

A lot of (passive and active) attacks have been developed on the server-aided RSA computation protocols (called RSA-S1 and RSA-S2) proposed by Matsumoto, Kato and Imai [1] (e.g., see, [7-9,11]). Matsumoto, Imai, Laih and Yen [4] also proposed two-phase versions of the basic protocols (called RSA-S1M and RSA-S2M) to gain resistance against passive attacks, mainly to counter the passive attack proposed by Pfitzmann and Waidner [8]. On the other hand, Quisquater and De Soete [2] and Kawamura and Shimbo [5] proposed rather different protocols for server-aided RSA computation. In these protocols, the secret exponent is decomposed using a fixed basis and the messages sent to the server are independent of the secret. Consequently, they are as secure against passive attacks as RSA but require too much computation and communication. Recently, Béguin and Quisquater [13] developed another protocol using the fast exponentiation algorithm due to Brickell, Gordon, McCurley and Wilson [14].

The purpose of this paper is to investigate various security issues on server-aided RSA computation protocols and provide possible improvements. Our attention is mainly focused on the two-phase protocols, RSA-S1M and RSA-S2M. We first show that these protocols in fact do not have any resistance against active attacks, contrary to the previous claim [10]. The proposed attacks seem applicable to any server-aided protocol for RSA computation, including the protocols in [2] and [5]. Thus it seems essential that the final result should be checked by the client before being sent to the server. We then present a server-aided protocol into which signature verification capability is integrated. As a result, the computation result can be efficiently verified in any server-aided RSA computation scheme, irrespective of the size of the public exponent.

Though the client only outputs the correct signature, there still exists another threat, the multi-round active attack such that the server deduces some partial information on the involved secret in each trial of attack by observing whether or not the client outputs its computation result (e.g., see [8,11,12]). We show that the parameter restrictions suggested in [11] are not effective against more sophisticated attacks. In fact, it seems impossible that every possible attack could be detected just by placing restrictions on the parameter selection. Thus we also assume that the secret exponent is newly decomposed in each run of the protocol and discuss some means of speeding up this preprocessing step.

The last security issue in these protocols is to prevent the implementation-dependent attack such as the server infers some information on the secret vectors by monitoring the duration of the client's computation. To avoid this type of attack, we have to require that the client should spend the same amount of time on each computation step or perform the required computation after receiving and storing all communications from the server. Under the latter condition, we propose a new method for decomposing the secret and performing each computation step fast. The resulting protocols are shown to be quite efficient and practical.

2 Proposed Active Attacks

In server-aided RSA computation protocols, the client wants to compute $y = x^d \bmod n$, where n is the product of two large primes p and q, with the aid of the powerful server. This section presents new active attacks on the protocols RSA-S1M and RSA-S2M proposed by Matsumoto et al. [4]. Similar attacks can be devised on the protocols in [2,5] as far as the computed signature is not checked.

One point needs to be mentioned on the passive attack. Unlike to the assertion in [4], the birthday-type attack [8] can be applied to the two-phase protocols as well. Of course, its complexity is much increased in this case, compared to single-phase protocols. But it is still much lower than that for a simple exhaustive search. For details, see the complexity analysis of Section 6.

2.1 Attack on RSA-S1M

The following protocol, RSA-S1M, is a two-phase version of RSA-S1 proposed to counter the passive attack devised by Pfitzmann and Waidner [8].

0) (Preprocessing) The client chooses an integer vector $\mathbf{D} = \{d_i\}_{i=1}^{M}$ over $\mathbf{Z}_n$ and binary vectors $\mathbf{F} = \{f_i\}_{i=1}^{M}$ and $\mathbf{G} = \{g_i\}_{i=1}^{M}$ such that $d = f \cdot g \bmod \lambda(n), f = \sum_{i=1}^{M} f_i d_i \bmod \lambda(n)$ and $g = \sum_{i=1}^{M} g_i d_i \bmod \lambda(n)$, where $\mathbf{F}$ and $\mathbf{G}$ have Hamming weight $\leq W$. The client also randomly picks an integer $r \in \mathbf{Z}_n$ and computes $t = r^{-g} \bmod n$.
1) The client sends $n, \mathbf{D}$ and x to the server.
2) The server returns $\mathbf{Z} = \{z_i\}_{i=1}^{M}$ such that $z_i = x^{d_i} \bmod n$.
3) The client computes and sends to the server $z = r \cdot \prod_{i=1}^{M} z_i^{f_i} = r \cdot x^f \bmod n$.
4) The server returns $\mathbf{V} = \{v_i\}_{i=1}^{M}$ such that $v_i = z^{d_i} \bmod n$.
5) The client computes the final result y as $y = t \cdot \prod_{i=1}^{M} v_i^{g_i} = t \cdot z^g \bmod n$.

Anderson [9] devised a simple one-round active attack on RSA-S1 and later Yen and Laih [10] claimed that the above two-phase protocol is highly resistant against Anderson's attack. However, we show that Anderson's attack can be extended to RSA-S1M (in fact, to any server-aided RSA computation scheme).

The server's strategy is to manipulate its transmissions in such a way that the final result y only consists of the product of known numbers. For this, the server supplies $z_i' = x^{ed_i} \bmod n$ in step 2) and $v_i' = p_i \cdot (z')^{d_i} \bmod n$ in step 4) respectively, where p_i's are small primes whose product is less than n. Now, when receiving $y' = t \cdot \prod_{g_i=1} v_i' \bmod n$ from the client, the server computes $y' \cdot x^{-1} = \prod_{g_i=1} p_i \bmod n$. The server can then factor this number and obtain the secret vector $\mathbf{G}$ (hence, the secret number g). Once g is obtained, the server can find the other secret vector $\mathbf{F}$ using the identity $x = (x^{eg})^f \bmod n$ in about about $N \log_2 N$ operations with $N = \sum_{k \leq \lceil W/2 \rceil} \binom{M}{k}$ [8]. This attack can also be applied to the non-binary version of the protocol if $\prod_{g_i \neq 0} p_i^{g_i}$ is less than n.

The above attack can be avoided if the client sends a blinded x in step 1) and cancels out the blinding factor in step 5). However, we can also devise a variation of the attack for this case : The server returns in step 4) $v_i' = \gamma_i \cdot z^{d_i} \bmod n$ with random γ_i and, when receiving y', computes $(y')^e \cdot x^{-1} = \prod_{g_i=1} \gamma_i^e \bmod n$. Now this equation can be used to find the secret vector $\mathbf{G}$ using the birthday paradox. Consequently, the whole attacking complexity in this case can be reduced to about two times $N \log_2 N$ operations with $N = \sum_{k \leq \lceil W/2 \rceil} \binom{M}{k}$.

2.2 Attack on RSA-S2M

The following is (non-binary) RSA-S2M, a variant of RSA-S1M which employs the Chinese remainder theorem to speed up the client's computation.

0) (Preprocessing) The client chooses integer vectors $\mathbf{D} = \{d_i\}_{i=1}^M$ ($d_i \in \mathbf{Z}_n$), $\mathbf{F_j} = \{f_{ji}\}_{i=1}^M$ and $\mathbf{G_j} = \{g_{ji}\}_{i=1}^M$ ($j = 1, 2$) such that $d = f \cdot g \bmod \lambda(n)$, $f = \sum_{i=1}^M f_{1i} d_i \bmod p-1$, $f = \sum_{i=1}^M f_{2i} d_i \bmod q-1$, $g = \sum_{i=1}^M g_{1i} d_i \bmod p-1$ and $g = \sum_{i=1}^M g_{2i} d_i \bmod q-1$, where the vectors $\mathbf{F_j}$'s and $\mathbf{G_j}$'s consist of small integers and their components satisfy $f_{1i} = f_{2i} \bmod 2$ and $g_{1i} = g_{2i} \bmod 2$ for all values of i. The client also randomly picks an integer $r \in \mathbf{Z}_n$ and computes $t = r^{-g} \bmod n$.
1) The client sends $n, \mathbf{D}$ and x to the sever.
2) The server returns $\mathbf{Z} = \{z_i\}_{i=1}^M$ such that $z_i = x^{d_i} \bmod n$.
3) The client computes $x^f \bmod n$ as $x^f = x_p^f w_p + x_q^f w_q \bmod n$, where $x_p^f = \prod_{i=1}^M z_i^{f_{1i}} \bmod p$, $x_q^f = \prod_{i=1}^M z_i^{f_{2i}} \bmod q$, $w_p = q(q^{-1} \bmod p)$ and $w_q = p(p^{-1} \bmod q)$. The client then sends $z = r \cdot x^f \bmod n$ back to the server.
4) The server returns $\mathbf{V} = \{v_i\}_{i=1}^M$ such that $v_i = z^{d_i} \bmod n$.
5) The client computes $z^g \bmod n$ as $z^g = z_p^g w_p + z_q^g w_q \bmod n$, where $z_p^g = \prod_{i=1}^M v_i^{g_{1i}} \bmod p$ and $z_q^g = \prod_{i=1}^M v_i^{g_{2i}} \bmod q$. The client then computes the final result y as $y = t \cdot z^g \bmod n$.

Shimbo and Kawamura [7] developed a factorization attack on RSA-S2, and claimed that their attack could be prevented by the parameter restriction such as adopted in step 0) in the above protocol (i.e., $f_{1i} = f_{2i} \bmod 2$ and $g_{1i} = g_{2i} \bmod 2$). However this is not true. We present a generalized version of the attack which can be applied to any server-aided RSA computation protocol using the Chinese remainder theorem (CRT).

The proposed attack is applied to the second phase of the protocol. Instead of returning $v_i = z^{d_i} \bmod n$ in step 4), the server sends back $v_i' = \gamma \cdot z^{d_i} \bmod n$ with γ chosen at random. Let $s_1 = \sum_{i=1}^M g_{1i}$ and $s_2 = \sum_{i=1}^M g_{2i}$) and, for the moment, assume that $s_1 \neq s_2$. Then the client's computation in step 5) will end up with the value y' given by

$$y' = t \cdot (\gamma^{s_1} z_p^g \cdot w_p + \gamma^{s_2} z_q^g \cdot w_q) \bmod n, \tag{1}$$

where $\gamma^{s_1} z_p^g$ and $\gamma^{s_2} z_q^g$ are actually numbers reduced mod p and mod q respectively. (Note, however, that the equation still holds even if they are not reduced.) By raising y' to the e-th power, the server obtains

$$(y')^e = t^e \cdot (\gamma^{es_1} z^{eg} + (\gamma^{es_2} - \gamma^{es_1}) z_q^{eg} w_q) \bmod n, \tag{2}$$

where we used the fact that $w_p^e = w_p \bmod n$, $w_q^e = w_q \bmod n$ and $w_p w_q = 0 \bmod n$. Therefore, on receiving y', the server can find the prime factor p by computing $\gcd(n, (y')^e - x\gamma^{es_1} \bmod n)$ since it is unlikely that q divides $t^e \cdot (\gamma^{es_2} - \gamma^{es_1}) z_q^{eg} \bmod n$. The server may try all small numbers within a reasonable bound as candidates for s_1 since it does not know the exact value of s_1. But s_1 will be small in practice and thus the prime factor can be quickly found.

The above attack cannot be prevented by the simple restriction on the secret vectors $\mathbf{G_1}$ and $\mathbf{G_2}$ such that $\sum_{i=1}^M g_{1i} = \sum_{i=1}^M g_{2i}$, since the server may use a set of random numbers as γ and compute v_i''s as above with a number randomly picked in the set. This will increase the number of gcd computations, but the attack will be still effective due to the small size of integers used. In fact, unless the two secret vectors $\mathbf{G_1}$ and $\mathbf{G_2}$ are the same, the proposed attack has a high probability of success only with several values of γ. However, setting $\mathbf{G_1} = \mathbf{G_2}$ and $\mathbf{F_1} = \mathbf{F_2}$ makes RSA-S2M essentially equivalent to RSA-S1M. Therefore, we conclude that server-aided RSA computation based on the CRT can hardly be secure against the active attack unless the client checks the final result.

3 Integrating Server-Aided Verification

The active attacks described seem applicable to any server-aided protocol for RSA computation. Thus it is essential that the client check the correctness of the final result before sending it to the server. If e is chosen to be small such as 3, the last checking will not much increase the client's computational load.

However, the RSA system with a small public exponent may be dangerous in some circumstances (e.g., when used for encryption in network environments, see [15]). Thus what is more desirable is that no restriction needs to be placed on the choice of e. It is then quite natural to consider that the verification of the computed signature could be carried out with the aid of the same server. Several such protocols were proposed (for example, see [2,16]), but they seem to require too much amount of communication or computation. Thus it remains still open to design an efficient verification protocol.

The problem of integrating signature verification capability is nontrivial. A main difficulty arises from the fact that the two processes are reverse each other (i.e., encrypting a signed message reveals the plaintext). This makes it difficult for the client to detect the server's attempt to obtain a signature on message of its own choosing. We here propose an efficient protocol for RSA signature generation with the correctness check. We only describe a variant of RSA-S1M and the same technique can be adapted for any SASC protocol. Suppose that $\gcd(3, \lambda(n)) = 1$ and $e > 3$.

0) (Preprocessing) The client computes d_0 such that $e_0 d_0 = 1 \bmod \lambda(n)$ with $e_0 = 3$ and randomly decomposes $d_0 d \bmod \lambda(n)$ as in RSA-S1M. It also computes $t_1 = r_1^{-g} \bmod n$ and $t_2 = r_2^{e_0 e} \bmod n$ with random $r_1, r_2 \in \mathbf{Z}_n$.
1) The client sends $n, \mathbf{D}$ and x to the server.
2) The server returns $\mathbf{Z} = \{z_i\}_{i=1}^{M}$ such that $z_i = x^{d_i} \bmod n$.
3) The client computes and sends to the server $z = r_1 \cdot \prod_{f_i=1} z_i \bmod n$.
4) The server returns $\mathbf{V} = \{v_i\}_{i=1}^{M}$ such that $v_i = z^{d_i} \bmod n$.
5) The client computes y_0 as $y_0 = t_1 \cdot \prod_{g_i=1} v_i = x^{d_0 d} \bmod n$. Then the client sends $v = y_0 \cdot r_2 \bmod n$ back to the server.
6) The server returns $w = v^e \bmod n$ to the client.
7) The client checks that $w^{e_0} = x \cdot t_2 \bmod n$. Only if the check succeeds, does the client send $y = y_0^{e_0} = x^d \bmod n$ to the server.

The client's on-line computational load increased due to the integration of signature verification is just six multiplications, irrespective of the size of e. This is a considerable advantage over direct verification if e is not very small. In step 7) the client may send y_0 directly to the server, which can then compute the signature y from y_0. Though this can save two multiplications, we would not like to recommand this variant since it may cause a problem in certain careless use of the protocol (see the footnote below). (It also requires that x^{d_0} mod n be of no meaning.)

To pass the check of step 7), the server has to know the e_0-th root of xt_2 mod n. This is infeasible if the server deviates the protocol. Note that d_0 and d are fixed but r_1 and r_2 are refreshed in each run of the protocol. This means that multi-round attacks on the signature verification part is of no use. Note also that the server may obtain the e_0-th root of t_2 (i.e., r_2^e mod n) by providing $z_i' = x^{e_0 d_i}$ mod n in step 2) and then computing $v^e x^{-1}$ mod n in step 6). However, there exists no way to obtain x^{d_0} mod n.[1] Thus we believe that the above protocol can detect any active attack mounted by the server.

4 Multi-Round Attacks Under Parameter Restriction

Even if the client checks the final result and only outputs the correct signature, there exists another threat to the server-aided protocol, i.e. the multi-round active attack such that the server changes a few values of its transmissions each time and decides, say parity or equality etc., on the corresponding elements of the secret vector based on whether or not the client gives the signature. Burns and Mitchell [11] described various attacks on RSA-S1 and RSA-S2, including multi-round active attacks, and proposed some means of parameter selection with which they claimed any attempt to deceive the client would be detected. However, this kind of parameter restriction seems not sufficient to detect every possible attack. As an example, we describe a new attack on RSA-S1. (This attack was in fact turned out to be a special case of the general attack described in [12], see also the footnote in the next page.)

In (non-binary) RSA-S1, d is simply decomposed as $d = \sum_{i=1}^{M} f_i d_i$ mod $\lambda(n)$ where f_i's are small positive integers. On request of the client, the server replies with $z_i = x^{d_i}$ mod n and the client then computes the signature y as $y = \prod_{i=1}^{M} z_i^{f_i}$ mod n. To counter Gollmann's attack (see [11] for details), Burns et al. proposed to restrict all f_i's to odd integers (hence no f_i is zero). Even under this restriction, the following attack can be successful.

The server randomly picks two integers, say j and k, in $[1, M]$ and sends back correct values for z_i for all values of i such that $i \neq j$ and $i \neq k$. In the latter two cases, the server provides $z_j = 2x^{d_j}$ mod n and $z_k = 2^{-1}x^{d_k}$ mod n. Then the client will get $y = 2^{f_j - f_k}x^d$ mod n and thus output it only if $f_j = f_k$.

[1] There is one thing to note when the client sends y_0 instead of y in step 7). If the client signs the same message x twice, then the server with knowledge of x^{d_0} mod n can obtain the signature on message x' of its own choosing. For this, the server returns $z_i' = (x')^{e_0 d_i}$ mod n in step 2), obtains $t_2^{d_0} = v^e(x')^{-1}$ mod n in step 6) and then replies with $w = (xt_2)^{d_0}$ mod n. Since this w passes the check of step 7), the server will get $y_0 = (x')^d$ mod n, the desired signature.

Therefore, by observing whether or not the client outputs the result, the server can determine whether f_j equals f_k. Similarly, in case that $f_j \neq f_k$, it can also determine whether $af_j = bf_k$ for some small odd integers a and b.

Repeating the above attack, the server can find all f_i's if it can meet the same client as many times as required. Or, using some partial information obtained from the attack, it can accelerate an exhaustive search. The only way to reduce this threat is to choose f_i's all distinct, but this is unacceptable in both efficiency and security. The parameter restriction on RSA-S2 seems more effective, but it can be easily seen that a similar strategy of comparing (in this case) two pairs of elements at a time can substantially reduce the security level. (It is possible to completely determine secret vectors for the binary case.)

Multi-round active attacks based on binary tests (even or odd, zero or not, equal or not, etc.) can be applied to any server-aided protocol as far as the involved secret vectors are fixed (see also [12]). [2] Thus the secret vectors should be randomly changed in each run (more practically, in limited runs) of the protocol. This will practically nullify any kind of active attacks on protocols using the random decomposition of the secret.

In this respect, protocols using a fixed basis, such as KS (Kawamura-Shimbo) and QS (Quisquater-De Soete) protocols, seem rather disadvantageous, in spite of their advantage of obvious security against passive attacks. One possible alternative to defeating multi-round active attacks in this kind of protocols is to construct two-phase variants, like RSA-S1M, using the decomposition $d = fg \bmod \lambda(n)$. In this case, one of f and g can be chosen arbitrarily small without loss of security, as far as it can provide large enough possibilities of decomposition. This will somewhat reduce the increase of complexity caused by two times serial execution of the original protocol. More preferably, d may be decomposed as $d = fg + h \bmod \lambda(n)$, where f and g are chosen to be of size $|\lambda(n)|/2$, and h is computed as $h = d - fg \bmod \lambda(n)$. Then h needs to be publicly transmitted.

5 Speeding up the Preprocessing Step

Assuming that the use of the same secret vectors is limited to a fixed times of protocol execution to counter multi-round active attacks, we need some means of speeding up the preprocessing step. Note that the client may use the same secret vectors repeatedly unless the failure count associated with the secret vectors exceeds a fixed threshold (say, five). However, it is clear that random blinding factors must be refreshed in each protocol execution.

[2] Kawamura [12] has already demonstrated the vulnerability of server-aided protocols to this kind of attack. He showed that in one of the KS protocols the secret exponent d could be deduced in 680 trials of attack for base $b = 16$ and, as a practical precaution, suggested that in order to limit the information leakage, the client refuse to interact with the same server if the final check fails more than a fixed threshold value. However, we have to note that although information leakage may be controlled by limiting the number of failures with the same server, a number of servers may conspire to find the secret of a specific client. Since partial information can add up to information sufficient to derive the secret, this precaution seems still unsatisfactory.

5.1 Precomputation of Blinding Factors

Let us first consider a method of accelerating the precomputation of r^{-g} mod n for random r and g. (We assume that r^{e_0e} mod n for random r is computed directly since e is not so large in most cases.) Note first that neither r nor $t = r^{-g}$ mod n is disclosed since they are used as $z = r \cdot x^f$ mod n and $y = t \cdot z^g$ mod n (see RSA-S1M). This means that r can be chosen at random in a restricted domain. One possible way, which we will adopt in this paper, is to use preprocessing algorithms for random exponentiation such as the one proposed by Schnorr [17,18]. For this, the prime factors p, q of n are chosen so that a prime β divides both $p-1$ and $q-1$ and then a base α of order β mod n is randomly generated. Now the client securely stores a small number of pairs $\{s, \alpha^s \bmod n\}$ with random $s \in \mathbf{Z}_\beta$. Then, during an idle time the client can compute and store several values of α^k mod n using a preprocessing algorithm. One of these values, say α^{k_r} mod n, can be used as r and another, say α^{k_t} mod n, as t. Then g is determined by $g = -k_r^{-1}k_t \bmod \beta$.

For security and performance of the above preprocessing method, several remarks need to be stated.

- Since r and t are never revealed and are refreshed in every protocol run[3], it seems unnecessary to choose β large. For example, β can be a prime of size $64 \sim 80$. Due to the same reason, de Rooij's attacks [19,20] on Schnorr's preprocessing algorithm cannot not be applied to our application.
- Due to the small order of α, one can easily find β if any power of α is known. If β is known, one can find f using $z^\beta = x^{\beta f}$ mod n. Thus it is essential to keep α and β secret. However, note that there exists no known algorithm for factoring n (for $|n| \geq 512$) faster with knowledge of β of size $64 \sim 80$.
- From the above remarks, we can see that Schnorr's algorithm can be safely used even with smaller sizes of security parameters than those given in [17]. Thus it is possible to compute one value of α^k mod n in around 10 multiplications mod n. For a practical implementation, all computations mod n can be performed mod p and mod q and then the results can be combined using the CRT. This will almost halve the computational amount.

5.2 Random Decomposition of the Secret

Since the secrecy of d does not rely on the randomness of d_i's, we may choose the integer vector $\mathbf{D}$ in some convenient ways. One attractive example (similar to Strategy C in [11]) is to decompose d as $d = cfg + h \bmod \lambda(n)$. Note that g is determined during the precomputation of r^{-g} mod n. Suppose the case of non-binary RSA-S1M (see Section 6.2 for the case of RSA-S2M). The client

1. generates random numbers $d_i (1 \leq i \leq M-1)$ of l-bit size (say, $l = 80$),
2. determines secret integer vectors $\mathbf{F} = \{f_i\}_{i=1}^M$ and $\mathbf{G} = \{g_i\}_{i=1}^M$,

[3] The server may obtain r by returning $z_i = 1$ for all values of i. Thus the client must check that $z \neq 1$ before sending z in step 3).

3. computes $d_M = (g - \sum_{i=1}^{M-1} g_i d_i) g_M^{-1} \bmod \beta$ (adds some multiples of β if $|\beta| < l$) and $f = \sum_{i=1}^{M} f_i d_i$ and
4. finally computes h as $h = d - cfg \bmod \lambda(n)$ with random $c \in [0, \lambda(n))$.

Here $d_i (1 \leq i \leq M)$, c and h are public numbers transmitted in step 1). To reduce the communication complexity, the client may generate $d_i (1 \leq i \leq M-1)$ and c from a small initial seed using a common pseudorandom number generator (e.g., a linear congruential generator). Then it will suffice for the client to send just the seed, d_M and h. Differences from the original RSA-S1M are that the server returns $z_i = x^{cd_i} \bmod n$ in step 2) and one more value $z_h = x^h \bmod n$ in step 4). (Note that z_h is needed at the end of step 5).)

This method of decomposition has several advantages : smaller number of pseudorandom bit generation, no inversion mod $\lambda(n)$ and dramatic reduction of the server's computation complexity. Note that evaluating many exponentials with the same base can be substantially speeded up by using precomputation [14,21]. However, this decomposition definitely gives up keeping about $|\lambda(n)| - 2l$ bits of information on d and thus may be subject to some attacks exploiting this fact. One possibility is to apply Wiener's attack [22] on short RSA secret keys using continued fractions. But this kind of attack seems not applicable to the above case. Note that if c is chosen in the interval $[0, \lceil \lambda(n)/fg \rceil)$, then Wiener's attack may be successful unless e and $L = \gcd(p-1, q-1)$ are large.

To see this, let $h = ch_q + h_r$ $(0 \leq h_r < c)$ and note that $ed = ec(fg + h_q) + eh_r = 1 + \frac{K}{L}(n - p - q + 1)$ where a common factor of K and L can be cancelled out. Let $A = nK + (1 - eh_r)L$. Then we get $\frac{ec}{A} = \frac{1}{(fg+h_q)L}(1 - \delta)$ with $\delta = \frac{K(p+q-1)}{A}$. Thus the continued fraction attack will be successful if $\delta < \frac{2}{3(fg+h_q)L}$. This condition holds as far as $|fgL|$ is a little less than $|n|/2$. Here we need to find the correct value of A by guessing $K (< e)$ and L. But e is usually small and L is also small if not intentionally chosen to be large. Thus one can try all small values of K and L within certain bounds and apply the continued fraction algorithm to find the prime factors of n. This analysis shows that we had better choose c over $[0, \lambda(n))$ rather than taking other measures. Note, however, that in our decomposition c may be chosen over $[0, \lceil \lambda(n)/fg \rceil)$ since we have already chosen L large enough to defeat the above attack. On the other hand, we can avoid most weaknesses which may potentially exist by setting $c = 1$ (i.e., $2l \simeq |\lambda(n)|$) at the expense of more computations.

6 Improving Performance of Non-binary Schemes

We now assume that the client checks the computation result and that the preprocessing phase is carried out in each run of the protocol. Then the last threat to server-aided protocols will be the implementation-dependent attack (e.g., see [11]). Suppose that the computation of (say) $z = \sum_{f_i=1} z_i \bmod n$ should be done in a step-by-step manner, considering the storage limitation of typical smart cards. That is, the server supplies one value of z_i at a time and the client multiplies it into the partial result. This type of processing makes it possible for the server to monitor the client's computation time to deduce the weights of f_i's. Thus, to avoid this attack, the client has to spend the same amount of time on each computation step or it can compute z after receiving and storing all z_i's at a time. We adopt the latter strategy and propose a method for improving the performance of non-binary schemes.

6.1 The Case of RSA-S1M

Suppose that d is decomposed as described in Section 5.2. Let $f_i = \sum_{j=1}^{K} 2^{j-1} f_{ij}$ and $g_i = \sum_{j=1}^{K} 2^{j-1} g_{ij}$ with $f_{ij}, g_{ij} \in \{0,1\}$. These K-bit integers are chosen so that the total binary weight of $\mathbf{F} = \{f_i\}_{i=1}^{M}$ and $\mathbf{G} = \{g_i\}_{i=1}^{M}$ respectively is at most W, i.e. Weight(**F**)=Weight(**G**) $\leq W$. Then the value of z in step 3) (similarly y in step 5) can be computed as

$$z = r \cdot \prod_{j=1}^{K} \left(\prod_{i=1}^{M} z_i^{f_{ij}} \right)^{2^{j-1}} \mod n. \tag{3}$$

For this, the client receives all z_i's at a time (hence it needs M temporary registers for them) and then performs the following algorithm.

```
receive and store all $z_i$'s;
$z := 1$;
for $j := K$ to 1 step -1
    $z := z^2 \mod n$;
    for each $i$, if($f_{ij} = 1$)
        $z := zz_i \mod n$;
$z := zr \mod n$;
```

It can be easily seen that the above algorithm can compute z in at most $K+W-1$ multiplications ($K-1$ squarings + W multiplications). Note that if the total weight of secret integers are chosen to be less than W, then we need some simulation of multiplications to prevent the implementation-dependent attack.

We now consider the computational complexity for finding the secret d. Let $\mathbf{S_f} = \{\sum_{i=1}^{M} f_i d_i | \text{Weight}(\mathbf{F}) \leq W\}$ and $\mathbf{S_g} = \{\sum_{i=1}^{M} g_i d_i | \text{Weight}(\mathbf{G}) \leq \lceil W/2 \rceil\}$. The most promising way to find d would be an exhaustive search based on the equation $1 - eh - ecfg^{\star} = ecfg^{\circ} \mod \lambda(n)$. That is, the server computes $x^{1-eh-ecfg^{\star}} \mod n$ and $x^{ecfg^{\circ}} \mod n$ for all possible values of $f, g^{\star}$ and g° such that $f \in \mathbf{S_f}$ and $g^{\star}, g^{\circ} \in \mathbf{S_g}$ and then searches for equality by sorting all these values. This will reveal the secrets f and g. Therefore, the computational complexity for this attack will be about $N \log_2 N$ operations where N, the total number of values to be sorted (disregarding a factor 2), is given by

$$N = \sum_{w=1}^{W} \binom{MK}{w} \sum_{w=1}^{\lceil W/2 \rceil} \binom{MK}{w}. \tag{4}$$

For a practical estimation of the performance, we have chosen the attacking complexity of 2^{72} operations. Thus we have to choose parameters so that $N \log_2 N \geq 2^{72}$ (i.e., $|N| \sim 66$). Note that for the above attack the server also needs a storage of order N. Table 1 summarizes the resulting performance for some small M's. The number of multiplications required of the client, $COMP$, is given by $2(K+W)-1$. The number of $|n|$-bit blocks to be communicated, $COMM$, is $2M+5$, where for simplicity we counted the seed, d_M and h altogether as two blocks (see Section 5.2). For these figures we did not take into account the final check. Thus if $e=3$, $COMP$ needs to be increased by 2. Or if the protocol in Section 3 is used, $COMP$ and $COMM$ should be increased by 6 and 2 respectively.

M	3	4	5	6	7	8	9
K	15	13	11	9	8	7	8
W	15	13	13	13	13	13	11
$COMM$	11	13	15	17	19	21	23
$COMP$	59	51	47	43	41	39	37

Table 1 : Selected parameters for RSA-S1M

The actual performance needs to be evaluated by considering the communication speed of the client (e.g., 9.6 Kbps for typical smart cards, but there exist smart cards with much faster interfaces, from 19.2 up to 115.2 Kbps) and the storage (RAM) available (typically 128 ~ 512 bytes). Note that the client may store M z_i's in EEPROM if it does not have a sufficient RAM space, since they need not be updated during the computation.[4] In this case the client has to write $2M$ $|n|$-bit numbers into EEPROM. Finally note that the server has to perform M exponentiations with l-bit exponents in each phase and two full size exponentiations (for c and h, see Section 5.2). The PC with a DSP accelerator card (e.g., see [23]) seems powerful enough as a practical server.

6.2 The Case of RSA-S2M

Applying the same technique to RSA-S2M, we can further reduce the client's computation time. For the sake of simplicity, we here assume that reduction of $|n|$-bit number to $\frac{|n|}{2}$-bit number and multiplication of two $\frac{|n|}{2}$-bit numbers take the same time. Then, even for the case where the client computes z and y using the CRT, it can be seen that the signature can be generated in the equivalent of $\frac{COMP+M}{2}+3$ multiplications mod n. Let us consider the performance of non-binary RSA-S2M in more details.

[4] Typical EEPROM has a byte-write time of 5 ms (2 ms with recent technology) and thus writing a 512 bit number consumes about 320 ms (128 ms, resp.). However, with somewhat complicated coding it is possible for the smart card to perform EEPROM writes in parallel with other operations.

The secret d can be decomposed as $d = fg + h \bmod \lambda(n)$ as follows. Without loss of generality, we assume that $p < q$. The client

1. generates $M-1$ random numbers $d_i (1 \le i \le M-1)$ of size $|n|/2$,
2. selects secret integer vectors $\mathbf{F_j} = \{f_{ji}\}_{i=1}^{M+1}$ and $\mathbf{G_j} = \{g_{ji}\}_{i=1}^{M}$ $(j = 1, 2)$ such that Weight$(\mathbf{F_1} \cup \mathbf{F_2}) \le W$ and Weight$(\mathbf{G_1} \cup \mathbf{G_2}) \le W$,
3. computes $d_M = (g_0 - g')g_{1M}^{-1} \bmod \beta + k\beta$ where g_0 is a value determined during precomputation of t, $g' = \sum_{i=1}^{M-1} g_{1i}d_i \bmod p-1$ and $k \in [0, p/\beta)$,
4. computes $d_{M+1} = g - \sum_{i=1}^{M} g_{2i}d_i \bmod q-1$ where $g = g' + d_M g_{1M} \bmod p-1$,
5. computes $d_{M+2} = f - \sum_{i=1}^{M+1} f_{2i}d_i \bmod q-1$ where $f = \sum_{i=1}^{M+1} f_{1i}d_i \bmod p-1$ and finally $h = d - fg \bmod \lambda(n)$.

The client can send $x, n, d_M, d_{M+1}, d_{M+2}, h$ and the seed used to generate $d_i (1 \le i \le M-1)$. The remaining part of the protocol should be clear from the above decomposition and the original protocol RSA-S2M.

To get the attacking complexity, let $\mathbf{S_g} = \{\sum_{i=1}^{M} g_{1i}d_i | \text{Weight}(\mathbf{G_1}) \le \lceil W/2 \rceil\}$ and $\mathbf{S_f} = \{\sum_{i=1}^{M+1} f_{1i}d_i | \text{Weight}(\mathbf{F_1}) \le \lceil W/2 \rceil\}$. The attacking server computes $\gcd(x^{egf^\star} - x^{1-eh-egf^\circ} \bmod n, n)$ for all possible values of g, $f^\star$ and f° : If Weight$(\mathbf{G_1}) \le \lceil W/2 \rceil$, then the server can find the prime factor p with $g \in \mathbf{S_g}$ and $f^\star, f^\circ \in \mathbf{S_f}$. Otherwise, i.e. if Weight$(\mathbf{G_2}) \le \lceil W/2 \rceil$, we have $g \in \{\mathbf{S_g} + d_{M+1}\}$, $f^\star \in \mathbf{S_f}$ and $f^\circ \in \{\mathbf{S_f} + d_{M+2}\}$ and thus the prime factor q can be found. Note that in either case $f^\star + f^\circ$ covers all possible values of f. The number of possible values that $x^{egf^\star} \bmod n$ can take is given by

$$N = \sum_{w=1}^{\lceil W/2 \rceil} \binom{MK}{w} \sum_{w=1}^{\lceil W/2 \rceil} \binom{(M+1)K}{w}. \tag{5}$$

This birthday-type attack needs about $N(\log_2 N)^2$ operations and a storage of order N (see Section 5.2 in [13]).[5]

M	3	4	5	6	7	8	9
K	11	9	8	7	7	6	5
W	21	19	19	17	17	17	17
$COMM$	16	18	20	22	24	26	28
$COMP$	26.5	24.0	23.5	22.0	22.5	22.0	21.5

Table 2 : Selected parameters for RSA-S2M

Table 2 shows the resulting performance of RSA-S2M for the complexity of 2^{72} operations (i.e., $|N| \sim 60$). Here $COMP$ denotes the equivalent number of

[5] A naive approach to performing the required gcd computations would require about N^2 operations. We are indebted to Béguin and Quisquater [13] for knowing that there exists an algorithm with complexity of $N(\log_2 N)^2$ operations.

multiplications mod n, evaluated under the previous assumption on multiplication and modular reduction, which can be shown to be $K + 0.5(M + W) + 3.5$. $COMM$ is computed as $2M + 10$ (counting the seed and d_M as one block). As for the three values, $z_{M+1} = x^{d_{M+1}} \bmod n$, $v_{M+2} = z^{d_{M+2}} \bmod n$ and $z_h = x^h \bmod n$, the client may request each of them at the time when it is needed, since they are used at the end of some computations. Thus it suffices for the client to store only M values during the computation of each phase.

7 Concluding Remarks

In this paper we have investigated various security issues on server-aided RSA computation protocols, mainly focused on the two-phase protocols, RSA-S1M and RSA-S2M, and provided possible improvements.

We described new one-round active attacks that can be applied to any protocol for server-aided RSA computation if the final result is not checked. As a practical countermeasure, we presented an efficient protocol for server-aided validation of the computed signature. This protocol allows the client to check the correctness of the computation result at most six modular multiplications, irrespective of the size of the public exponent. We also showed that the preprocessing step should be carried out in each protocol execution to counter multi-round active attacks and discussed possible means of speeding up such preprocessing. Finally, using a new method for selecting (secret and public) parameters, we proposed modifications of RSA-S1M and RSA-S2M and analyzed their performance. The resulting protocols seem to be quite efficient.

There may be a slight disadvantage in two-phase protocols : the requirement of precomputation. Though precomputation for each signature generation is not much expensive (e.g., about the equivalent of 10 multiplications mod n, see Section 5.1), the client has to store a certain (predetermined) number of precomputed values so that several signatures can be generated successively without time delay due to precomputation. If this is still undesirable in practical applications, we may use a modification of RSA-S2 by the same technique, in which such precomputation is unnecessary. However, in this case the computational load of the client will be somewhat increased, compared to RSA-S2M. The protocol proposed by Béguin and Quisquater [13] has some advantage over two-phase protocols in this connection, since it also uses no precomputation. But it can be seen that using our proposed technique will give better efficiency in both computation time and storage usage than using the algorithm in [14].

Acknowledgement : The authors would like to thank Philippe Béguin, Tsutomu Matsumoto and Shin-ichi Kawamura for providing some corrections and helpful comments on an earlier version of this paper.

References

1. T.Matsumoto, K.Kato and H.Imai, Speeding up secret computations with insecure auxiliary devices, In *Proc. of Crypto'88*, Springer-Verlag, LNCS 403, 497-506 (1990).

2. J.J.Quisquater and M.De Soete, Speeding up smart card RSA computation with insecure coprocessors, In *Proc. Smart Card 2000*, North-Holland, 191-197 (1991).
3. C.S.Laih, S.M.Yen and L.Harn, Two efficient server-aided secret computation protocols based on addition chain sequence, In *Proc. of Asiacrypt'91*, S.V., LNCS 739, 450-459 (1993).
4. T.Matsumoto, H.Imai, C.S.Laih and S.M.Yen, On verifiable implicit asking protocols for RSA computation, In *Proc. of Auscrypt'92*, S.V., LNCS 718, 296-307 (1993).
5. S.Kawamura and A.Shimbo, Fast server-aided secret computation protocols for modular exponentiation, *IEEE JSAC*, 11(5), 778-784 (1993).
6. S.Kawamura and A.Shimbo, Performance analysis of server-aided secret computation protocols, *Trans. IEICE*, 73(7), 1073-1080 (1990).
7. A.Shimbo and S.Kawamura, Factorization attack on certain server-aided secret computation protocols for the RSA secret transformation, *Elect. Lett.*, 26(17), 1387-1388 (1990).
8. B.Pfitzmann and M.Waidner, Attacks on protocols for server-aided RSA computation, In *Proc. of Eurocrypt'92*, S.V., LNCS 658 (1993).
9. R.J.Anderson, Attack on server-aided authentication protocols, *Elect. Lett.*, 28(15), 1473 (1992).
10. S.M.Yen and C.S.Laih, More about the active attack on the server-aided secret computation protocol, *Elect. Lett.*, 28(24), 2250 (1992).
11. J.Burns and C.J.Mitchell, Parameter selection for server-aided RSA computation schemes, *IEEE Trans. Computers*, 43(2), 163-174 (1994).
12. S.Kawamura, Information leakage measurement in a distributed computation protocol, *IEICE Trans. Fundamentals*, E78-A(1), 59-66 (1995).
13. P.Béguin and J.J.Quisquater, Fast server-aided RSA signatures secure against active attacks, In *this proceedings*.
14. E.F.Brickell, D.M.Gordon, K.S.McCurley and D.B.Wilson, Fast exponentiation with precomputation, In *Proc. of Eurocrypt'92*, S.V., LNCS 658, 200-207 (1993).
15. J.Hastard, On using RSA with low exponent in a public key network, In *Proc. of Crypto'85*, S.V., LNCS 218, 403-408 (1986).
16. T.Matsumoto, K.Kato and H.Imai, How to ask and verify oracles for speeding up secret computations (Part 2), *IEICE TR, IT89-24* (1989).
17. C.P.Schnorr, Efficient identification and signatures for smart cards, In *Proc. of Crypto'89*, S.V., LNCS 435, 239-252 (1990).
18. C.P.Schnorr, Efficient signature generation by smart cards, *J. Cryptology* 4 (3), 161-174 (1991).
19. P.de Rooij, On the security of the Schnorr scheme using preprocessing, In *Proc. of Eurocrypt'91*, S.V., LNCS 547, 71-78 (1991)
20. P.de Rooij, On Schnorr's preprocessing for digital signature schemes, In *Proc. of Eurocrypt'93*, S.V., LNCS 765, 435-439 (1994).
21. C.H.Lim and P.J.Lee, More flexible exponentiation with precomputation, In *Proc. of Crypto'94*, S.V., LNCS 839, 95-107 (1994).
22. M.J.Wiener, Cryptanalysis of short RSA secret exponents, *IEEE Trans. Inform. Theory*, IT-36, 553-558 (1990).
23. S.R.Dusse and B.S.Kaliski Jr., A cryptographic library for the Motorola DSP 5600, In *Proc. of Eurocrypt'90*, S.V., LNCS 473, 230-244 (1991).