

Short Paper

A Secure Server-Aided RSA Signature Computation Protocol for Smart Cards

GWOB OA HORNG

*Institute of Computer Science
National Chung-Hsing University
Taichung, Taiwan 402, R.O.C.
E-mail: gbhorng@cs.nchu.edu.tw*

Smart cards have opened up possibilities for many exciting applications. However, one problem with conventional smart cards is that they only have very limited computational power. As a result, it takes too long for a smart card to perform a single RSA signature operation in real time applications. Server-aided RSA signature computation protocols offer feasible solutions for this problem. The basic idea is to distribute most of the computation to an auxiliary processor which is capable of performing fast multi-precision modular exponentiation. However, the smart card has to guard against the auxiliary processor since it may attempt to obtain information about the secret exponent or to obtain the smart card's signature on a message of its own choosing by supplying the smart card with incorrect values. The only way to defeat these attacks is for the smart card to have some means of verifying the data provided by the auxiliary processor. In this paper, we propose such a secure protocol.

Keywords: smart card, digital signature, server-aided computation, RSA, active attack

1. INTRODUCTION

A smart card, also known as an IC card, appears similar to a bank card but contains a silicon chip. This gives it the capacity to process information internally and store all transactions in non-erasable memory. This has opened up exciting possibilities for its use with electronic money. However, conventional smart cards have very limited computational power. As a result, it takes too long for a smart card to perform a single digital signature operation.

A digital signature scheme is a protocol that produces the same effect as a real signature [1]. The most popular one is the RSA scheme [2]. To set up the RSA scheme, a user, say Alice, chooses two large primes p and q at random. Let $N = pq$ and $\phi(N) = (p - 1)(q - 1)$. Two more integers e and d are chosen such that $ed \equiv 1 \pmod{\phi(N)}$. The pair (e, N) is made publicly known. However, d , called the secret exponent, is kept secret. The digital signature created by Alice for a message M , $0 < M < N$, is the integer C such that $C \equiv M^d \pmod{N}$. Anyone can verify the correctness of the signature by checking whether M is congruent to C^e modulo N or not.

Received December 30, 1998; revised August 17, 1999; accepted January 3, 2000.
Communicated by Chi Sung Laih.

Therefore, creating a digital signature for a message based on the RSA scheme involves modular exponentiation. Since modular exponentiation takes a great deal of time to compute, it is inappropriate for a smart card with only limited computational power to perform RSA signatures in real time applications.

In 1988, the concept of server-aided RSA computation protocols was proposed [3]. The basic idea is to distribute most of the computation to an auxiliary processor which is capable of performing fast multi-precision modular exponentiation.

We will review the basic server-aided RSA computation protocol and address its potential security weakness in section 2. In section 3, a probabilistic data verification scheme is proposed. Security and performance analysis is also given. In section 4, we present a new server-aided RSA signature computation protocol based on the data verification scheme described in section 3. Finally, some concluding remarks are given in section 5.

2. SERVER-AIDED RSA SIGNATURE COMPUTATION PROTOCOLS

A server-aided RSA signature computation protocol is designed to make use of an auxiliary processor to enable a smart card to perform RSA signature computations faster. Of course, if the auxiliary processor were trusted, then we could give the secret exponent to it and ask it to perform the whole signature computation. However, this is not normally the case. Therefore, the design goal of server-aided RSA signature computation protocols is to make use of an auxiliary processor without compromising the secrecy of the RSA secret exponent.

Many such protocols have been proposed. According to whether or not the data transmitted between the auxiliary processor and the card is dependent on the secret exponent, they can be classified into two types: dependent protocols and independent protocols [4, 5]. The dependent protocols have better speedup compared to the independent ones. However, they are weaker in terms of security against passive attacks [6-8]. In this paper, we will only consider the dependent protocols.

Throughout this paper, let $N = pq$ be the modulus of the RSA scheme, let d be the secret exponent, and let M be the message to be signed. The fundamental idea of a dependent protocol [9] is to represent d as

$$d \equiv f_1 d_1 + \dots + f_s d_s \pmod{\phi(N)},$$

where f_i are in the range $[0, 2^c - 1]$ for some positive integers c and s . The values N , M and d_i , $1 \leq i \leq s$, are sent to the auxiliary processor. Then the auxiliary processor computes the values $z_i \equiv M^{d_i} \pmod{N}$, $1 \leq i \leq s$, and returns them to the card. Finally, the smart card computes $M^d \equiv \prod_{i=1}^s (z_i)^{f_i} \pmod{N}$. Many variants have been proposed, including those in [7, 10-12]. The number of bits of security against brute force search is cs . That is, the auxiliary processor requires 2^{cs} trials to deduce d .

However, all server-aided secret computation protocols are vulnerable. A malicious auxiliary processor can obtain the smart card's signature on a message X of its own choosing through the following naive attack: The auxiliary processor simply supplies the smart card with the values $X^{d_i} \pmod{N}$ instead of the requested values $M^{d_i} \pmod{N}$, for all i .

Moreover, a malicious auxiliary processor may try to obtain information about the secret exponent d through passive or active attacks. In a passive attack, the auxiliary processor sends the correct values of $M^{d_i} \pmod{N}$, $1 \leq i \leq k$, back to the card. Nevertheless, it

tries to search for the secret exponent based on the parameters provided by the card [6-9]. For example, a divide-and-conquer attack can effectively reduce the number of bits of security by approximately 50% [9]. In an active attack, the auxiliary processor sends incorrect values of $M^{d_i} \bmod N$, $1 \leq i \leq k$, back to the card and tries to deduce the secret exponent d .

The card can choose large values of s and c to increase the search space in order to counter passive attacks [9]. However, guarding against active attacks is more difficult. One way is for the smart card to verify the final signature. One such approach, suggested in [13], is to choose a small value of e . In [9], it is assumed that the public exponent is 3. Therefore, the smart card can check the signature before disclosing it. Indeed, this control can effectively exclude all one-round attacks, such as those in [13, 14]. However, since $e = 3$ is fixed, the choices of p and q are restricted to primes r such that $3r - 1$. This is not a serious problem. Nevertheless, application of such an RSA scheme may be limited. For example, it has been shown that sending linear related messages to different users in an RSA system is insecure when the public exponent is small [15]. Furthermore, multi-round active attacks [6] are still possible. For example, the server can decide whether f_i is odd [9], whether $af_i = bf_j$ [7], or whether $f_i = r$ [16], for the indices i, j and integers a, b, r . In fact, many protocols which were claimed to be secure have been shown to be insecure under multi-round active attacks [17-19].

To counter multi-round active attacks, Lim and Lee proposed that the same set of $\{d_1, d_2, \dots, d_s\}$ is limited to a fixed number of protocol executions [7]. They also proposed a method for accelerating the precomputation. However, this was shown to be insecure in [20]. We believe the most effective way to guard against active attacks is for the smart card to verify the correctness of the data provided by the auxiliary processor. In the next section, we will propose a probabilistic data verification scheme for verifying the data provided by the auxiliary processor.

3. A PROBABILISTIC DATA VERIFICATION SCHEME

Of course, the smart card can not recompute the modular exponentiation by itself and compare the result with the data provided by the auxiliary processor. Nevertheless, it can send a request to the auxiliary processor to compute an additional modular exponentiation for the purpose of verification. The basic idea is as follows: Suppose we want the auxiliary processor to compute $M^k \bmod N$. Then, we can ask it to compute $M^k \bmod N$ and $(aM^b)^k \bmod N$ by transmitting the values M_1, M_2, N , and k to it, where $M_1 \equiv M \bmod N$, $M_2 \equiv aM^b \bmod N$, and the values of a and b are randomly chosen [21]. It is impossible for the auxiliary processor to compute b while knowing only M_1 and M_2 since for every possible value b' of b , there is an a' such that $M_2 \equiv a'M_1^{b'} \bmod N$.

Let a be randomly chosen from $[1 .. N - 1]$, and let b be randomly chosen from $[0..2^c - 1]$ for some constant c . Suppose Z_1 and Z_2 are the data received from the auxiliary processor claimed to be M_1^k and M_2^k , respectively. It is easy to see that if both Z_1 and Z_2 are correct, then $Z_1^b \equiv a^{-k} Z_2 \bmod N$.

Suppose $Z_1 \not\equiv M_1^k \bmod N$. Let $Z_1 \equiv yM_1^k \bmod N$, where $y \not\equiv 1 \bmod N$. Since $Z_1^b \equiv a^{-k} Z_2 \bmod N$, Z_2 must be congruent to $y^b a^k M^{kb} \bmod N$. The auxiliary processor can randomly generate an integer for Z_2 . However, the probability that $Z_2 \equiv y^b a^k M^{kb} \bmod N$ is only $1/(N-1)$. Since the auxiliary processor can know $y, a^k M^{kb}$ from Z_1, M_1, M_2 , and k , it can do better than randomly generate an integer for Z_2 . However, to compute the value $y^b a^k M^{kb}$

mod N for Z_2 , the auxiliary processor needs to guess the right value of b or of a . Since b is randomly chosen from $[0..2^c - 1]$, the probability of guessing the right value of b is $1/2^c$. Similarly, the probability of guessing the right value of a is $1/(N - 1)$. Therefore, if $Z_1^b \equiv a^{-k} Z_2 \pmod{N}$, then $Z_1 \equiv M_1^k \pmod{N}$ and $Z_2 \equiv M_2^k \pmod{N}$ with probability $1 - 1/2^c$. That is, the probability that the auxiliary processor can succeed with an active attack is only $1/2^c$.

Let us analyze the cost of this verification scheme. Suppose the smart card uses a square-and-multiply algorithm to compute $M^b \pmod{N}$ [22]. Then, it needs to perform $l - 1$ squaring and w multiplications, where l is the length of the binary representation of b and w is the number of ones in the binary representation of b . Hence, the total number of modular multiplications for computing $M^b \pmod{N}$ is at most $2c - 1$. There are more efficient algorithms for implementing modular multiplication over smart cards [23].

Therefore, setting up the data verification scheme, that is, computing $aM^b \pmod{N}$, requires at most $2c$ modular multiplications. Assume that the value of $a^{-k} \pmod{N}$ is precomputed and stored in the smart card. Then, verifying the correctness of the received data, that is, checking $Z_1^b \equiv a^{-k} Z_2 \pmod{N}$, also requires at most $2c$ modular multiplications.

However, there are two problems with this scheme. One is that the cost may be too high for the scheme to be practical. The other is that once the auxiliary processor guesses the right value of b , it can determine the value of the fixed parameter a . As a result, other active attacks can no longer be detected. We can solve the first problem by taking advantage of the fact that a modular multiplication mod p takes somewhere between one-fourth and one-half of the time required for a modular multiplication mod N [9]. The second problem can be solved by introducing another random integer into M_2 . Note that $p|N$. The details are as follows.

To ask the auxiliary processor to compute $M^k \pmod{N}$:

- Step 1. The smart card randomly chooses integers a , r from $[1 .. N - 1]$ and b from $[0..2^c - 1]$ for some constant c .
- Step 2. The smart card sends k , N , M_1 and M_2 to the auxiliary processor, where $M_1 = M$ and $M_2 \equiv ((aM^b \pmod{p}) + rp) \pmod{N}$.
- Step 3. The auxiliary processor computes Z_1 , Z_2 and sends them back to the smart card, where $Z_1 \equiv M_1^k \pmod{N}$ and $Z_2 \equiv M_2^k \pmod{N}$.
- Step 4. The smart card verifies the correctness of Z_1 , Z_2 by checking $Z_1^b \equiv a^{-k} Z_2 \pmod{p}$.

In the following, we will discuss the security and complexity of the scheme. First, we will show that if the auxiliary processor is honest, then $Z_1^b \equiv a^{-k} Z_2 \pmod{p}$. This is demonstrated by the following congruences:

$$\begin{aligned}
 a^{-k} Z_2 \pmod{p} &\equiv (a^{-k} M_2^k \pmod{N}) \pmod{p} \\
 &\equiv (a^{-k} (((aM^b \pmod{p}) + rp) \pmod{N})^k \pmod{N}) \pmod{p} \\
 &\equiv (a^{-k} (aM^b)^k) \pmod{p} \\
 &\equiv M^{bk} \pmod{p} \\
 &\equiv (M^k)^b \pmod{p} \\
 &\equiv Z_1^b \pmod{p}
 \end{aligned}$$

Then we will show that the auxiliary processor can not determine a or p from M_1 and M_2 . Since $M_2 \equiv ((aM^b \bmod p) + rp) \bmod N$, it is possible that M_1 and M_2 may reveal useful information to the auxiliary processor. Let $\alpha = M^b \bmod p$. Then, $M_1 = \alpha + r_1p$ and $M_2 = a\alpha + r_2p$ for some integers r_1 and r_2 . There are at least three unknowns (namely, a , α , p) in these two equations. Therefore, the auxiliary processor can determine neither a nor p from them. Nevertheless, if $a \equiv 1 \bmod p$, then the auxiliary processor can compute p since $p | \gcd(N, M_1 - M_2)$. However, since a is randomly chosen from $[1 .. N - 1]$, the probability that $a \equiv 1 \bmod p$ is very small (about $(N/p)/N = 1/p$).

Suppose the auxiliary processor is malicious. Instead of computing $Z_1 \equiv M_1^k \bmod N$, it computes $Z_1' \equiv X^k \bmod N$ for some $X \neq M_1$. To cheat successfully, Z_2' must satisfy the following equation: $Z_1'^b \equiv a^{-k} Z_2' \bmod p$. If the auxiliary processor randomly generates an integer for Z_2' , then the probability that $Z_1'^b \equiv a^{-k} Z_2' \bmod p$ is only $1/p$. Since p is very large, the probability is very small. However, if the auxiliary processor can guess correctly the value of b , then it can successfully cheat the client by sending Z_2' to the client such that $Z_2' \equiv Z_2 (XM^{-1})^{bk} \bmod N$. Since b is randomly chosen from $[0..2^c - 1]$, the probability of guessing the right value of b is $1/2^c$.

Finally, we will show that a remains secure even if b is guessed. Since the auxiliary processor does not know the value of p , there are two unknowns a and r in the equation $M_2 \equiv ((aM^b \bmod p) + rp) \bmod N$. That is, due to the additional random integer r and the unknown p , the value of a remains secure even the auxiliary processor correctly guesses the value of b .

Therefore, the above modified scheme has the same security level as the original one under active attacks. That is, if $Z_1^b \equiv a^{-k} Z_2 \bmod p$, then $Z_1 \equiv M_1^k \bmod N$ and $Z_2 \equiv M_2^k \bmod N$ with probability $1 - 1/2^c$.

In the following, we will show that the computation cost is reduced. Setting up the data verification scheme, that is, computing $((aM^b \bmod p) + rp) \bmod N$, requires at most $2c$ modular multiplications mod p and one modular multiplication mod N . Assume that the value of $a^k \bmod p$ is precomputed and stored in the smart card. Then, verifying the correctness of the received data, that is, checking $Z_1^b \equiv a^{-k} Z_2 \bmod p$, requires at most $2c$ modular multiplications mod p . Since a modular multiplication mod p takes somewhere between one-fourth and one-half of the time required for a modular multiplication mod N [9], the new scheme is more efficient than the original scheme.

4. A SECURE SERVER-AIDED RSA SIGNATURE COMPUTATION PROTOCOL

In this section, we will propose a new server-aided RSA signature computation protocol based on the data verification scheme described in the previous section. The protocol includes the following preprocessing phase:

- The smart card randomly generates s integers f_1, f_2, \dots, f_s from the range $[0..2^{c_1} - 1]$ such that f_s and $\phi(N)$ are relative prime. Then, it randomly generates $s - 1$ integers d_1, d_2, \dots, d_{s-1} from the range $[1..N - 1]$ and computes $d_s \equiv f_s^{-1} (d - (\sum_{i=1}^{s-1} f_i d_i)) \bmod \phi(N)$. Therefore, $d \equiv f_1 d_1 + \dots + f_s d_s \bmod \phi(N)$.
- The smart card randomly chooses s integers a_i from the range $[1..N - 1]$ and computes $A_i \equiv a_i^{-d_i} \bmod p$. The values f_i, d_i, a_i, A_i , for $i = 1, 2, \dots, s$, are stored in the smart card.

To sign a message M , the smart card performs the following steps:

- Step 1. The smart card randomly generates s integers b_1, b_2, \dots, b_s , from $[0 .. 2^{c_2} - 1]$ for some predetermined constant c_2 and another s integers r_1, r_2, \dots, r_s , from $[1 .. p - 1]$.
- Step 2. The smart card computes $M_i \equiv ((a_i M^{b_i} \bmod p) + r_i p) \bmod N$. Then, the smart card sends $N, M, M_1, M_2, \dots, M_s, d_1, d_2, \dots, d_s$ to the auxiliary processor.
- Step 3. The auxiliary processor computes the pairs (U_i, V_i) , for $i = 1, 2, \dots, s$, and sends them back to the smart card, where $U_i \equiv M^{d_i} \bmod N$ and $V_i \equiv M_i^{d_i} \bmod N$.
- Step 4. The smart card verifies the correctness of the received data by checking whether or not $U_i^{b_i} \equiv A_i V_i \bmod p$, for $i = 1, 2, \dots, s$.

If it is correct, then go to Step 5. Otherwise, the smart card terminates the protocol and performs the signature computation by itself.

- Step 5. The smart card computes the digital signature S , where $S \equiv \prod_{i=1}^s U_i^{f_i} \bmod N$.

Notice that the random numbers $b_i, 1 \leq i \leq s$, are not fixed. They are generated independently each time the smart card initiates the protocol. That is, an independent set of b_i is generated for each run. Furthermore, the protocol stops (Step 4) when it suspects that the auxiliary processor is malicious (that is, the verification fails). Therefore, no matter how many times the protocol has executed, the probability of guessing the right value of b remains $1/2^c$.

Let us analyze the time complexity of the protocol. There are three sources of costs: the operations performed by the smart card (including the generation of random numbers, data verification, and the computation of the final signature), the communication overhead between the smart card and the auxiliary processor, and the exponential operations performed by the auxiliary processor. They are discussed in the following.

The smart card needs to generate $2s$ random numbers, namely, b_i, f_i , for $i = 1, 2, \dots, s$. To verify the correctness of the data, the smart card must perform $2c_2(s + 1)$ modular multiplications, mod p , and one modular multiplication, mod N . In [9], it is proved that the expected number of modular multiplications (MMs) for computing the exponentiation $M^x \bmod N$ is $3|x|/2 - 5/2 + 2^{1-|x|} < 3|x|/2$, where $|x|$ is the size of x . Therefore, computation of the final signature requires that $(s - 1) + 3c_1s/2$ modular multiplications mod N be performed.

Next, we will consider the communication overhead. The smart card needs to send $2s + 2$ numbers, namely, N, M, M_i, d_i , for $i = 1, 2, \dots, s$, to the auxiliary processor, and the auxiliary processor needs to send $2s$ numbers, namely, U_i, V_i , for $i = 1, 2, \dots, s$, to the smart card.

Finally, the auxiliary processor must perform $2s$ modular exponentiations. The expected number of total modular multiplications, mod N , is $3s|N|$. The costs are summarized in the following table.

Smart card			Comm.	Aux. processor Exponential operations
Generating random integers	Data verification	Signature computation		
$2s$ (numbers)	$2c_2(s+1)$ $(+1 \bmod N)$ (mod p MMs)	$(s-1)+$ $3c_1s/2$ (mod N MMs)	$(4s+2) N $ (bits)	$3s N $ (mod N MMs)

The proper choices of the values of s , c_1 , and c_2 depend on the security requirement, the communication speed between the smart card and the auxiliary processor, and the computation power of the smart card and the auxiliary processor. However, it is better to choose a small value for s in order to minimize the communication overhead and the verification costs. Therefore, $s = 1$ is the best choice. Another advantage in choosing $s = 1$ is that no divide-and-conquer attack [9] is possible.

We can estimate the performance of the protocol based on the following values: $s = 1$, $c_1 = 64$ (that is, the number of bits of security is 64), and $c_2 = 5$ (that is, the probability that the auxiliary processor can succeed with an active attack is $1/32$). Furthermore, we assume that the smart card can compute m modular multiplications, mod N , per second and $2m$ modular multiplications, mod p , per second. The auxiliary processor is assumed to be 100 times faster than the smart card. We also assume that the communication speed is 9600 bit/s (the standard ISO interface), and that time required for the smart card to generate a random number is equal to the time it needs to perform a modular multiplication. Then, computing a signature takes $109/m + 6 \times |N|/9600 + 3 \times |N|/(100 \times m)$ seconds using a server-aided protocol and $1.5 \times |N|/m$ seconds without using a server-aided protocol. Therefore, for $|N| = 512$ and $m = 33$ [24], the speedup is $23.27/4.09 = 5.69$. If $|N| = 1024$ and $m = 33$, the speedup will be $46.55/4.87 = 9.56$.

5. CONCLUDING REMARKS

Smart cards have opened up possibilities for many exciting applications. Many of them require that smart cards perform digital signature operations. However, the problem with conventional smart cards is that they only have very limited processing power while, the RSA signature scheme is computationally intensive. Hence, performing a digital signature operation on a smart card takes too much time. Nevertheless, smart card readers can easily be equipped with more sophisticated processors capable of performing multi-precision modular exponentiation. Therefore, server-aided RSA signature computation offers a feasible solution to this problem.

However, a smart card has to guard against an auxiliary processor that attempts to obtain information about the secret exponent d or to obtain the smart card's signature on a message of its own choosing by supplying the smart card with incorrect values. The only way to defeat these attacks is for the smart card to have some means of validating the data provided by the auxiliary processor. In this paper, we have proposed such a protocol based on a probabilistic data validation scheme. It can detect active attacks mounted by an auxiliary processor with very high probability. Furthermore, the performance of the protocol can be improved by using the Chinese Remainder Theorem [9].

Modular exponentiation is probably the operation most frequently performed in cryptography. Our verification scheme can also be applied to other types of server aided modular exponentiation computation.

There is a tradeoff between security and speedup when using a server-aided protocol. High speedup will reduce the level of security. Which parameters should be chosen depends on the application. However, if high security is required, then one should use crypto-smart-cards [25] instead of conventional smart cards.

ACKNOWLEDGMENT

The author is grateful to the referees for their valuable comments.

REFERENCES

1. C. Pfleeger, *Security in Computing*, Prentice-Hall, 1989.
2. R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, Vol. 21, No. 2, 1978, pp. 120-126.
3. T. Matsumoto, K. Kato, and H. Imai, "Speeding up secret computations with insecure auxiliary devices," in *Advances in Cryptology: CRYPTO'88*, 1988, pp. 497-506.
4. J. Quisquater and M. D. Soete, "Speeding up smart card RSA computation with insecure coprocessors," in *Smart Card 2000*, D. Chaum, Ed., Amsterdam: North-Holland, 1991, pp. 191-197.
5. S. Kawamura and A. Shimbo, "Fast server-aided secret computation protocols for modular exponentiation," *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 5, 1993, pp. 778-784.
6. B. Pfitzmann and M. Waidner, "Attacks on protocols for server-aided RSA computation," in *Advances in Cryptology: Eurocrypt'92*, Lecture notes in computer science, Vol. 658, 1993, pp. 153-162.
7. C. Lim and P. Lee, "Security and performance of server-aided RSA computation protocols," in *Advances in Cryptology: CRYPTO'95*, Lecture notes in computer science, Vol. 718, 1995, pp. 70-83.
8. J. Merkle and R. Werchner, "On the security of server aided RSA protocols," *Electronic Colloquium on Computational Complexity*, TR97-027, 1997.
9. J. Burns and C. Mitchell, "Parameter selection for server-aided RSA computation schemes," *IEEE Transactions on Computers*, Vol. C-43, No. 2, 1994, pp. 163-174.
10. C. Laih, S. Yen, and L. Harn, "Two efficient server-aided secret computation protocols based on the addition sequence," in *Advances in Cryptology- ASIACRYPT'91*, 1993, pp. 450-459.
11. P. Belguin and J. Quisquater, "Fast server-aided RSA signatures secure against active attacks," in *Advances in Cryptology: CRYPTO'95*, Lecture notes in computer science, Vol. 718, 1995, pp. 57-69.
12. S. Hwang, C. Chang, and W. Yang, "Two efficient server-aided RSA secret computation protocols against active attacks," *IEICE Transactions Fundamentals*, Vol. E79, No. 9, 1996, pp. 1504-1511.
13. A. Shimbo and S. Kawamura, "Factorization attack on certain server-aided computation protocols for the RSA secret transformation," *Electronics Letters*, Vol. 26, No. 17, 1990, pp. 1387-1388.
14. R. Anderson, "Attack on server assisted authentication protocols," *Electronics Letters*, Vol. 28, No. 15, 1992, pp. 1473.
15. J. Hastad, "Solving simultaneous modular equations of low degree," *Journal of SIAM Computing*, Vol. 17, No. 2, 1988, pp. 336-341.
16. G. Horng, "An active attack on protocols for server-aided RSA signature computation,"

- Information Processing Letters*, Vol. 65, No. 2, 1998, pp. 71-73.
17. G. Horng, "Active attacks on two efficient probabilistic server-aided RSA secret computation protocols," *IEICE Transactions Fundamentals*, Vol. E80-A, No. 10, 1997, pp. 2038-2039.
 18. S. Kawamura, "Information leakage measurement in a distributed computation protocol," *IEICE Transactions Fundamentals*, Vol. E78, 1995, pp. 59-66.
 19. P. Nyugen and J. Stern, "The Belguin-Quisquater server-aided RSA protocol from crypto'95 is not secure," in *Proceedings of Advances in Cryptology – ASIACRYPT'98*, K. Ohta and D. Pei (Eds.), Vol. 1514 of Lecture Notes in Computer Science, pp. 372-379.
 20. J. McKee and R. Pinch, "Further attacks on server-aided RSA cryptosystems," to appear.
 21. G. Horng, "A probabilistic data verification scheme for server-aided RSA signature computation protocols," in *Proceedings of International Conference on Cryptology and Information Security*, 1996, pp. 23-26.
 22. D. Stinson, *Cryptography: Theory and Practice*, CRC press, 1995.
 23. C. Koc, T. Acar, and B. Kaliski, Jr., "Analyzing and comparing montgomery multiplication algorithms," *IEEE Micro*, Vol. 16, No. 3, 1996, pp. 26-33.
 24. J. Dhem, D. Veithen, and J. Quisquater, "SCALPS: Smart card for limited payment systems," *IEEE Micro*, Vol. 16, No. 3, 1996, pp. 42-51.
 25. D. Naccache and D. M'Raihi, "Cryptographic smart cards," *IEEE Micro*, 1996, Vol. 16, No. 3, pp. 14-24.

Gwoboa Horng (洪國寶) received the B.S. degree in Electrical Engineering from National Taiwan University in 1981, and the M.S. and Ph.D. degrees from the University of Southern California in 1987 and 1992, respectively, all in Computer Science.

Since 1992, he has been on the faculty of the Institute of Computer Science at National Chung-Hsing University, Taichung, Taiwan. His current research interests include artificial intelligence, computer security, cryptography, and symbolic computation.