Speeding Up Secret Computations with Insecure Auxiliary Devices

Tsutomu MATSUMOTO
Koki KATO
Hideki IMAI

Division of Electrical and Computer Engineering
YOKOHAMA NATIONAL UNIVERSITY
156 Tokiwadai, Hodogaya, Yokohama, 240 Japan

Abstract This paper deals with and gives some solutions to the problem of how a small device such as a smart card can efficiently execute secret computations using computing power of auxiliary devices like (banking-, telephone-, ...) terminals which are not necessarily trusted. One of the solutions shows that the RSA signatures can be practically generated by a smart card.

1. Introduction

Small devices such as smart cards or IC cards are easy to be carried and have the ability to compute, memorize, and protect data. Such convenient *ultimate personal computers*[1] have been useful tools for constructing various information systems and now they are expected to be utilized in much wider applications. Unfortunately, smart cards now available are not so powerful, still the jobs we want them to execute are liable to hard for them. For example, many want to realize public key cryptographic algorithms in smart cards. But these are not easy tasks for them. Even if future smart cards would be more powerful, the gap would not be filled, because we would require them more intelligence.

An easy and usual way to overcome this situation is the use of auxiliary computers such as (network-, POS-, banking-, telephone-, facsimile-, ...) terminals for supplying the short-computing power of smart cards. In the following we use the terms '*client*' and '*server*'. A *client* denotes the main device such as a smart card which has a secret computation and can execute the computation by itself but takes long time because of the lack of computing resources. A *server* denotes the auxiliary device which has enough computing resources.

If a *server* is trustworthy and will not leak the secrets, the *client* can pass the *server* the description of the secret computation and can ask the *server* to perform it and to tell the result. As an example, in a public key signature scheme, a *client* sends

S. Goldwasser (Ed.): Advances in Cryptology - CRYPTO '88, LNCS 403, pp. 497-506, 1990.

the message to be signed and the secret key to a *server* for generating a signature on behalf of the *client*.

But *servers* are not always trustworthy. A terminal in a public telephone booth or a POS terminal in a supermarket, etc., may be a *server* which may be equipped with a wiretapping device or might be infected by some computer viruses. When the *server* is insecure, the *client* has to protect its secret from the *server* during the interaction.

How a *client* can securely accelerate secret computations by using untrustworthy *servers* ? This is the problem to be solved in this paper. We believe this problem will be very important for our future's daily life. In our prior paper [2], we have pointed out the importance of the problem and presented some primal considerations. In the following sections we demonstrate several protocols solving the problem for (1) matrix computation, (2) modular equations, and (3) the RSA cryptosystem.

2. Related Works and Assumptions

There are some other researches [3][4][5][6][7][8] looking like ours.

Privacy Homomorphisms, proposed by Rivest-Adleman-Dertouzos [3] and recently examined by Brickell-Yacobi [4] and by Ahituv-Lapid-Neumann [5], are cryptographic functions preserving some operations. For example, when two data a and b are stored in a database in the form of ciphertexts $f(a)$, $f(b)$, if the enciphering function f is homomorphic with respect to an operation $\circ$ in the domain of f and an operation $\bullet$ in the codomain of f, then the ciphertext of the data $a \circ b$ can be obtained as $f(a) \bullet f(b)$ without deciphering and re-enciphering.

Similar notion called the *Directly Transformed Link Encryption* has been proposed by Matsumoto-Okada-Imai [6] in the field of network security. In each node of a communication network with the link encryption, each ciphertext c comming from an input link i is deciphered into a plaintext $m = D_i(c)$ and then, after a routing, enciphered into another ciphertext $c' = E_j(m)$ to be emitted into an output link j. Here D_i and E_j are deciphering and enciphering algorithms associated with the input link i and the output link j, respectively. The core idea of the directly transformed link encryption is to use instead of D_i and E_j an algorithm H_{ij} which directly transforms c into c' so that the security of the plaintext in the node is enhanced. Cryptosystems based on power functions are examples for those applicable to the directly transformed link encryption.

Though the notions of privacy homomorphism and the directly transformed link encryption are attractive, they don't suffice for our porposes.

On the other hand, Feigenbaum [7] and Abadi-Feigenbaum-Kilian [8] studied the problem of so-called *Computing with Encrypted Data*. Their problem is very similar to ours. However, their stance seems to be a little bit different from ours. Since their interest was focused on the theory, if we use our terminology, they assumed that the *client* has probabilistic polynomial time computing power and that the *server* has unlimited computing power and they derived interesting conclusion that hard functions are also difficult to be securely encrypted. Their work is very interesting but not sufficient to our practical problem.

To make clear the differences, we summarize here our assumptions specifically:

Assumptions A computation originally owned by a *client* is a feasible one with respect to an ordinary computer. In typical situations, it is probabilistic polynomial time computable. However, when the size of the input is small and then the computation is tractable by some ordinary computer, the computation might be outside the probabilistic polynomial time. *Servers* adopted for the speed up are such ordinary computers. That is, there are limitations to the computational complexity with which the *servers* can cope. *Servers* might leak the data treated in interactions with a *client*, but do not refuse the jobs given by the *client* nor send false answers to the *client*. Each *client* is sufficient to protect its secret from resource bounded enemies. But the computational complexity for that purpose used by the *client* should not beyond the computational complexity for executing the secret computation by itself.

3. General Idea of Speed Up

Let a client want to obtain the value $y = g(x)$ of a computable function g on input x. Assume that there is an algorithm (circuit) C_g to compute g which is practically tractable by a *server* but not by the *client*.

Our general idea of speed up is as follows.

Protocol P [9]

(0) The *client* randomly decompose the algorithm C_g into three algorithms (circuits) I, M, F such that (i) the consecurive applications of I and M and F compute the function g and (ii) the *client* can execute I and F in enough speed. For many practical applications, it is worth while considering a simplar version such that the *client* is restricted to select M from a predetermined set of algorithms.

(1) The *client* applies I to have $u = I(x)$, and sends $[M, u]$ to the *server*.

(2) The *server* applies M to u and sends $v = M(u)$ back to the *client*.

(3) The *client* obtains y by applying F to v as $y = F(v)$.

Efficiency and Security: Let $Comm(\alpha)$, $Comp_C(\beta)$, and $Comp_S(\gamma)$ denote the time to transfer α between the *client* and the *server*, the time to execute algorithm β in the *client*, and the time to execute algorithm γ in the *server*, respectively. The total time to execute steps (1),(2),(3) in the protocol P is

$$T(\mathrm{P}) = Comp_C(I) + Comm([M, u]) + Comp_S(M) + Comm(v) + Comp_C(F).$$

Thus the speed up effect of P is $Comp_C(C_g)/T(\mathrm{P})$. Given upper bound B_C of the computing time of the *client*, the security of the protocol P is roughly measured by the ambiguity

$$A(\mathrm{P}) = \#\{[I, F] | Comp_C(I) + Comp_C(F) \leq B_C\}.$$

Variations: The protocol P can be generalized into two directions by decomposing M further. One is to have a series of algorithms and adopt more interactions. The other is to have a set of algorithms executable in parallel by independent *servers*.

4. Demonstrating the Speed Up Protocols

We show now some of the basic protocols to demonstrate our idea of enhancing smart cards.

4.1 Speed Up via Coordinate Permutation

In usual computing environments, multiplying matrices is recognized as rather light computation. But it is not so easy for today's smart cards. On the other hand, permuting rows and columns of matrices can be performed by only changing indices of the components of matrices. We have verified through an experiment [2] that this technique actually works well for programs in smart cards. The following three examples are based on this fact. The coordinate permutation is a very useful tool for constructing speed up protocols.

[a] Matrix Multiplications

Target: A *client* has two secret matrices A and B and wants to obtain the product $C = AB$.
Assumption: The *client* can efficiently permute rows and columns of matrices. There is a *server* which can multiply matrices overwhelmingly faster than the *client* can.

Protocol MM

(0) The *client* randomly generates permutation matrices P, Q and R.
(1) The *client* permutes A and B along with $[P, Q]$ and $[Q^{-1}, R]$ to have $A' = PAQ$, $B' = Q^{-1}BR$, and sends $[A', B']$ to the *server*.
(2) The *server* computes and sends back to the *client* $C' = A'B'$.
(3) The *client* obtains C by permuting C' along with $[P^{-1}, R^{-1}]$ as $C = P^{-1}C'R^{-1}$.

Remark: This protocol can be applied to the speed up of evaluating a tuple of multivariate polynomials.

[b] Linear Equations

Target: A *client* has a secret non-singular matrix A and a secret matrix B and wants to obtain the solution X of the equation $AX = B$.
Assumption: The *client* can efficiently permute rows and columns of matrices. There is a *server* which can solve linear equations overwhelmingly faster than the *client* can.

Protocol LE

(0) The *client* randomly generates permutation matrices P, Q and R.
(1) The *client* permutes A and B along with $[P, Q]$ and $[P, R]$ to have $A' = PAQ$, $B' = PBR$, and sends $[A', B']$ to the *server*.
(2) The *server* solves the equation $A'X' = B'$ for X' and sends X' back to the *client*.
(3) The *client* obtains X by permuting X' along with $[Q, R^{-1}]$ as $X = QX'R^{-1}$.

Remark: A slightly modified version of this protocol can be applied to the *linear programming* problem.

Evaluation: We have organized software experiments for Protocol MM and LE. The conclusion is that these protocols are effective since, as described above, the permutations can be done faster than the matrix multiplications and the amount of communication between the *client* and the *server* is only three matrices. However, by these protocols, the *server* might acquire some statistical information. Indeed, they cannot protect values of the functions not affected by permutations. An example is the determinant of C in Protocol MM. But we think there are many practical applications to which these protocols are useful.

[c] Graph Isomorphisms

Target: A *client* has two secret graphs a and b of the same number of verteces and edges with their adjacency matrices A and B, respectively, and wants to know whether these graphs are isomorphic or not. And if they are isomorphic, the *client* also wants to obtain the isomorphism, which is the permutation x of coordinates determined by the permutation matrix X satisfying the equation $AX = XB$.
Assumption: The *client* can efficiently permute rows and columns of matrices. There is a *server* which can quickly solve the graph isomorphism problem. But the *server* may not quickly solve the graph nonisomorphism problem.

Protocol GI

(0) The *client* randomly generates permutations p and q, to which correspond permutation matrices P, Q, respectively.
(1) The *client* permutes rows and columns of A and B along with p and q to have $A' = PAP^{-1}$, $B' = QBQ^{-1}$, and sends $[A', B']$ to the *server*.
(2) The *server* tries to decide , in a period of time, whether the graphs with adjacency matrices A' and B' are isomorphic or not. If they are decided to be isomorphic, the *server* computes the permutation x' corresponding to the isomorphism and sends x' to the *client*. Otherwise, the *server* sends '#' to the *client*.
(3) If x' is sent, the *client* obtains x by transforming x' with p^{-1} and q as $x = p^{-1}x'q$. If '#' is sent, the *client* decides that that a and b are not isomorphic.

Remark: This protocol can be applied to the speed up of *pattern recognition* based on the graph isomorphisms.

4.2 Modular Equations

Univariate polynomial equations over finite fields can be solved by polynomial time algorithms. And as Rabin [10] shows, there are efficient probabilistic algorithms for them. Main jobs of these algorithms are to take the greatest common divisor of two polynomials by applying the well known extended Euclidean algorithm. However, these are not so easy tasks for ordinary smart cards.

Target: A *client* has secret integers k and $a_0, a_1, a_2, \ldots, a_{m-1}$ such that the modular equation

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_{m-1} x^{m-1} + x^m \equiv 0 \pmod{k}$$

is solvable in x, and wants to obtain a solution x of this equation.
Assumption: The *client* can efficiently execute multiplication mod$k$ and division mod$k$. There is a *server* which can solve modular equations overwhelmingly faster than the *client* can.

Protocol ME

(0) The *client* randomly selects an integer r such that $\gcd(r, k) = 1$.
(1) The *client*and computes $[b_0, b_1, b_2, \ldots, b_{m-1}]$ by

$$c_0 = 1, \text{ and } c_i = r c_{i-1} \bmod k, \quad b_{m-i} = c_i a_{m-i} \bmod k \text{ for } i = 1, \ldots, m$$

and sends $[b_0, b_1, b_2, \ldots, b_{m-1}, k]$ to the *server*.
(2) The *server* obtains a solution y of the equation

$$b_0 + b_1 y + b_2 y^2 + \cdots + b_{m-1} y^{m-1} + y^m \equiv 0 \pmod{k}$$

and sends back y to the *client*.
(3) The *client* obtains x as $x = y r^{-1} \bmod k$.

Remark: This protocol can be generalized to fit any system of multivariate polynomial equations over any commutative ring.

Evaluation: Protocol ME is very effective, because in the protocol the computation the *client* has to do is only $2m$ multiplications mod$k$ and one division mod$k$ and the amount of communication between the *client* and the *server* is only $m + 2$ integers while the *server* could investigate nothing on the secret of the *client*.

5. Speeding Up the RSA Transformations

Is it possible to securely implement the RSA cryptosystem[11] with smart cards and terminals ? We think the answer is *yes*. For the RSA public transformation, a speed up protocol is described in [2]. For the RSA secret transformation, we show below two of the developed protocols.

Target: A *client* has integers x, d, n and wants to obtain the integer $y = x^d \bmod n$. The integer d is the secret of the *client*, while the integers n and e such that $ed \equiv 1 \pmod{\lambda(n)}$ are made public. Here n is the product of two large secret primes p, q $(p \neq q)$, and $\lambda(n)$ is the secret integer $\mathrm{lcm}(p-1, q-1)$.

For simplicity, the integer x may be known to the *server*. (It is an easy task to modify the following protocols with slightly adding the complexity so that x is also hidden from the *server*.)

5.1 Secret Powering 1

Assumption: The *client* can execute several multiplications mod$n$. There is a *server* which is equipped with a device which can implement the RSA secret transformation overwhelmingly faster than the naked *client* can.

Protocol RSA-S1

(0) The *client* randomly generates an integer vector $D = [d_1, d_2, \ldots, d_M]$ and a binary vector $F = [f_1, f_2, \ldots, f_M]$ such that

$$d \equiv f_1 d_1 + f_2 d_2 + \cdots + f_M d_M \pmod{\lambda(n)}$$

and $1 \leq d_i < n$ and $Weight(F) = \sum_{i=1}^{M} f_i \leq L$, where M and L are some integers.

(1) The *client* sends n, D, and x to the *server*.

(2) The *server* computes and sends back to the *client* $Z = [z_1, z_2, \ldots, z_M]$ such that

$$z_i = x^{d_i} \bmod n.$$

(3) The *client* obtains y by computing $y = y_M$ as follows:

$$y_0 = 1, \quad y_i = y_{i-1} z_i \bmod n \ \text{ if } \ f_i = 1; \quad y_i = y_{i-1} \ \text{ if } \ f_i = 0, \quad (i = 1, 2, \ldots, M).$$

Variation: We have a more general protocol if we exclude '*binary*' from the condition to F.

Complexity: Since the step (0) can be precomputed, for each x it is sufficient for the *client* to do at most $L - 1$ multiplications mod n. The amount of communication is $2(M + 1)$ integers of size at most $\log n$ bits.

Security: If the RSA cryptosystem is secure, the protocol could be broken only by searching true d via the exhaustion of

$$\sum_{i=1}^{L} \binom{M}{i} > \binom{M}{L}$$

possibilities.

Remark: If e is hidden from the *server*, Protocol RSA-S1 is applicable also to the case where $\lambda(n)$ can be readily computed from n. Secret powering over a finite field is an example.

Though such property are not preserved, we can have a more efficient protocol by utilizing the Chinese Remainder Theorem:

5.2 Secret Powering 2

Assumption: The *client* can execute several multiplications mod$p$ and mod$q$. The *client* has computed integers w_p and w_q such that

$$w_p = q(q^{-1} \bmod p), \quad w_q = p(p^{-1} \bmod q).$$

There is a *server* which is equipped with a device which can implement the RSA secret transformation overwhelmingly faster than the naked *client* can.

Protocol RSA-S2

(0) The *client* randomly generates an integer vector $D = [d_1, d_2, \ldots, d_M]$ and two binary vectors $F = [f_1, f_2, \ldots, f_M]$ and $G = [g_1, g_2, \ldots, g_M]$ such that

$$d \equiv f_1 d_1 + f_2 d_2 + \cdots + f_M d_M \pmod{p-1}$$

$$d \equiv g_1 d_1 + g_2 d_2 + \cdots + g_M d_M \pmod{q-1}$$

and $1 \leq d_i < n$ and $Weight(F) + Weight(G) = \sum_{i=1}^{M} f_i + \sum_{j=1}^{M} g_j \leq L$, where M and L are some integers.

(1) The *client* sends n, D, and x to the *server*.

(2) The *server* computes and sends back to the *client* $Z = [z_1, z_2, \ldots, z_M]$ such that

$$z_i = x^{d_i} \bmod n.$$

(3) The *client* obtains y by computing y as follows:

$$y = (y_{pM} w_p + y_{qM} w_q) \bmod n,$$

$$y_{p0} = 1, \quad y_{pi} = y_{p,i-1} z_i \bmod p \ \text{ if } \ f_i = 1; \quad y_{pi} = y_{p,i-1} \ \text{ if } \ f_i = 0,$$

$$y_{q0} = 1, \quad y_{qi} = y_{q,i-1} z_i \bmod q \ \text{ if } \ g_i = 1; \quad y_{qi} = y_{q,i-1} \ \text{ if } \ g_i = 0,$$

for $i = 1, 2, \ldots, M$.

Variation: We have a more general protocol if we exclude '*binary*' from the condition to F and G.

Complexity: Since the step (0) can be precomputed, for each x the amount of commputation the *client* has to do is equivalent to at most $3L/2$ multiplications mod p or mod q. The amount of communication is $2(M+1)$ integers of size at most $\log n$ bits.

Security: If the RSA cryptosystem is secure, the protocol could be broken only by searching true d via the exhaustion of

$$\sum_{j=1}^{L} \sum_{i=0}^{j} \binom{M}{i} \binom{M}{j-i} > \binom{M}{L/2}^2$$

possibilities.

Examples: Using a $i8086$ $(5MHz)$ (30 *msec* / 256-*bit* modular multiplication) or Z-80 $(6MHz)$ (300 *msec* / 256-*bit* modular multiplication) as a smart card with a single 64-*Kbps* (non-contact type) or 9600-*bps* serial link and the RSA hardwares (chips) [13] with speed 32-*Kbps* or 4800-*bps*, Protocol RSA-S2 can be accomplished about 4 to 30

times faster than the case where the microprocessor does the whole computation with the method due to Quisquater-Couvreur [12] (see Table A).

6. Conclusion

With several demonstrating examples, we have presented an important research problem of how to supply short-computing power of smart cards. The described protocols are all very simple but can be actually utilized in a system consists of smart cards and auxiliary computers. Other protocols and problems to be developed are described in [9].

Acknowledgment

The authors wish to thank Susumu Inomata for fruitful discussions and Kenji Koyama for providing Ref[13]. They also thank many who take interests in this work and anonymous refrees for their sugessions to improve this paper. Part of this work was supported by the Ministry of Education, Science and Culture under Grant-in-Aid for Encouragement of Young Scientists #62750283 and #63750316.

Table A. Examples of Protocol RSA-S2
[Processing time for a 512-bit message block]

	i8086 (5MHz)	Z80 (6MHz)
Serial Link 64Kbps + RSA hardware 32Kbps	L=20, M=50 1.7 sec <13.5 times faster>	L=12, M=142 7.7 sec <30 times faster>
Serial Link 9600bps + RSA hardware 4800bps	L=32, M=37 5.5 sec <4.2 times faster>	L=18, L=58 14.4 sec <16 times faster>
conventional method [12]	23 sec	230 sec

Modulus n : 512 bit Security $> 10^{20}$

References

[1] Svigals,J., *Smart Cards: The Ultimate Personal Computer*, Macmillan, 1985.

[2] Matsumoto,T., Kato,K. and Imai,H., "Smart cards can compute secret heavy functions with powerful terminals," (*written in Japanese with an English abstract*) *Proc. of 10th Symposium on Information Theory and Its Applications*, Enoshima-Island, Japan, pp.17-22, Nov.19-21, 1987.

[3] Rivest,R., Adleman,L. and Dertouzos,M., "On databanks and privacy homomorphisms," *Foundations of Secure Computation*, Demillo,R.A. *et al.*, editors, Academic Press, pp.168-177, 1978.

[4] Brickell,E.F. and Yacobi,Y., "On privacy homomorphisms," *Advances in Cryptology - EUROCRYPT'87*, Chaum,D. and Price,W.L. editors, Springer-Verlag, pp.117-125, 1988.

[5] Ahituv,N., Lapid,Y. and Neumann,S., "Processing encrypted data," Communications of the ACM, Vol.30, No.9, pp.777-780, Sep. 1987.

[6] Matsumoto,T., Okada,T. and Imai,H., "Directly transformed link encryption," (*in Japanese*) Trans. of IECE Japan, Vol.J65-D, No.11, pp.1443-1450, Nov. 1982.

[7] Feigenbaum,J., "Encrypting problem instances, or, ..., Can you take advantage of someone without having to trust him ? " *Advances in Cryptology - CRYPTO'85*, Williams,H.C. editor, Springer-Verlag, pp.477-488, 1986.

[8] Abadi,M., Feigenbaum,J. and Kilian,J., "On hiding information from an oracle," to appear in *Journal of Computer and System Sciences.* An extended abstract appeared in *Proc. of 19th Symposium on Theory of Computation*, pp.195-203, May, 1987.

[9] Matsumoto,T. and Imai,H., "How to use servers without releasing privacy – Making IC cards more powerful – ," (*in Japanese*) *IEICE Technical Report (ISEC)*, Vol.88, No.33, pp.53-59, May 1988.

[10] Rabin,M.O., "Probabilistic algorithms in finite fields," *SIAM J. Comput.*, Vol.9, No.2, pp.273-280, May 1980.

[11] Rivest,R., Shamir,A. and Adleman,L., "A method of obtaining digital signatures and public key cryptosystems," *Comm. of ACM*, Vol.21, No.2, pp.120-126, Feb. 1978.

[12] Quisquater,J.J. and Couvreuer,C., "Fast decipherment algorithm for RSA public-key cryptosystem," *Electron. Lett.* Vol.18, No.21, pp.905-907, Oct. 1982.

[13] Koyama,K., Table 1.(Developments of hardwares for the RSA cryptosystem), in "Information Security for Communications," (*in Japanese*) to appear in *Journal of the Institute of Television Engineers of Japan*, Dec. 1988.