A Smart Card Implementation of the Fiat-Shamir Identification Scheme

Hans-Joachim Knobloch

Institut für Algorithmen und Kognitive Systeme

Universität Karlsruhe (TH)

D-7500 Karlsruhe, FR Germany

Abstract

This paper describes results and experiences gained from the test implementation of an interactive identification scheme. It was intended to exploit the feasibility of an asymmetric crypto protocol for a state-of-the-art smart card environment. For that reason the identification scheme proposed by Fiat and Shamir was implemented between an actual smart card microprocessor and an industry standard personal computer with a smart card interface. The limits of a current smart card processor in terms of volatile and nonvolatile memory capacity and instruction set turned out to be a rather strict limitation for the choice of the algorithms used. The most time consuming task during the protocol is modular multiplication. Due to the processor structure it is performed as separate multiplication and reduction, where reduction is led back to integer multiplication. The current implementation allows the authentication of a 120 byte identification string at a security level of 2^{-20} within an average time of about 6 seconds. The experiences gained during this implementation led to a set of requirements for a future specialised processor for asymmetric cryptographic protocols that will be needed to increase this performance by some orders of magnitude.

C.G. Guenther (Ed.): Advances in Cryptology - EUROCRYPT '88, LNCS 330, pp. 87-95, 1988.

I. Introduction

During the last years, with the forthcoming of the commercial use of smart cards, some cryptographic protocols based on asymmetric ciphers have been proposed to use smart cards for identification, signatures, as electronic wallet etc. One may note that nearly all commercially available smart card systems use, if at all, only symmetric block ciphers, as asymmetric protocols are considered too complex for current smart card processors.

The Fiat-Shamir identification scheme is one of the simplest of the above mentioned asymmetric protocols as it does not need large amounts of stored data nor extensive communication or many protocol steps and it is therefore one of the most suitable for a test implementation on a smart card system.

II. The Processor

The smart card used in our project has an 8-bit microprocessor with 256 byte RAM and 2K byte E^2PROM (Electrically Erasable Programmable ROM) on chip for nonvolatile storage of data and program. Therefore the processor could be reprogrammed by the personal computer which was also its partner for the protocol. Thus several algorithms could be tested without having to wait for the production of a new ROM mask.

The ISO draft standard on identification cards [3] requires that all communication is done serially using only one contact pin for both input and output. Since the processor doesn't have a serial I/O unit the communication had to be implemented in software and thus needed code space and computing time. The mentioned draft standard includes a parity-generation-, parity-checking- and error-retry-protocol for the bidirectional I/O line. In order to save space for the protocol code and data, only a simple 9600 baud serial communication without parity generation was implemented.

The chip card processor's instruction set is similar to that of any conventional 8-bit microprocessor. Relevant details are an 8–times–8–to–16–bit multiplication instruction, requiring about 5 times the execution time of an 8-bit addition, whereas the instruction to program one byte into the E^2PROM, requires about 3300 times the execution time of an 8-bit addition. To gain better performance the latter fact implies that intermediate results have to be stored in RAM, but not in E^2PROM.

III. The Scheme

For a detailed discussion of the Fiat-Shamir identification scheme the reader is referred to the original publication [1], we will give a short review of this technique with emphasis on the particularities of the implementation.

The center issueing the cards chooses a public modulus n as the product of two secret primes p and q. For reasons explained below the implementation requires

$$2^{512} > n \geq 2^{512} - 2^{256}.$$

Now be

I a 960 bit (120 byte) ID-string of a user applying for a card,

$j \in [0,2^{16})$ and

e_i the ith 48 bit unit vector.

The center forms for $i = 1, 2, .. 48$

$$t_i = 2^{976} e_i + 2^{16} I + j,$$

$$u_i = \lfloor t_i / 2^{512} \rfloor \oplus (t_i \bmod 2^{512})$$

(where $\oplus$ means bitwise addition modulo 2),

$$v_i = \lfloor u_i / 2^{256} \rfloor \oplus (u_i \bmod 2^{256}) \text{ and}$$

$$w_i = f(v_i)$$

(where $f(k)$ means enciphering a fixed 512 bit plaintext with a block cipher with key k).

The term j is used to ensure that w_i is a quadratic residue mod n for at least 20 distinct values of i. For simplicity of notation from now on it will be assumed that these values of i are 1, ... , 20.

For $i = 1, ... , 20$ the center computes a

square root s_i of w_i (mod n)

using the knowledge of p and q and applying the chinese remainder theorem.

The card is personalized by storing

s_i for $i = 1, ... , 20$ and

$$P = 2^{976} \left(\sum_{i=1}^{20} e_i\right) + 2^{16} I + j$$

An identification device knows n and how to compute the w_i's from P.

The identification protocol between a smart card S and an identification device PC is:

1. S sends P to PC.
2. PC computes the w_i's.
3. S picks a pseudo random number $r' \in [0,2^{256})$, sets $r = 2^{256}r'$ and sends $x = r^2 \bmod n$ to PC.
4. PC sends to S a (pseudo) random binary vector $c = (c_1, .., c_{20})$.
5. S sends to PC:

$$y = r\prod_{c_i=1} s_i \mod n.$$

6. PC accepts P if and only if:

$$y^2 = x\prod_{c_i=1} w_i \mod n$$

If a forger guesses the vector c, he may send a value

$$\frac{r^2}{\prod_{c_i=1} w_i} \mod n$$

instead of x in step 3 and r instead of the product in step 5. The probability that PC accepts P if S doesn't know the s_i's is 2^{-20} (assuming equidistribution probability for c), if S performs only polynomial time computations and cannot compute in polynomial time a square root mod n of any product of some w_i's or their reciprocals. The proof for this statement is almost identical to the proof in Fiat's and Shamir's publication.

Remarks:

1. Since its inversion includes a known plaintext attack on the involved block cipher, the function used to compute the w_i's from the ID-string I should be strong enough to prevent a potential attacker from computing an ID-string out of known square roots modulo n.

2. Fiat's and Shamir's original protocol requires to use the multiplicative inverse of the s_i's on the smart card side. The check on step 6 of the protocol would then be, if

$$x = y^2\prod_{c_i=1} w_i \mod n$$

Using the s_i's rather than s_i^{-1} makes it possible that PC performs only one modular multiplication at step 6 of the protocol instead of two. The other multiplication can be done while the smart card still computes y. As the smart card will usually be the slower partner in the protocol, this fact slightly speeds up the overall execution time. However, if the inverse s_i's have to be used on the card side for some other reasons, only changes of the PC's program, not of the smart card's would be required.

3. The original protocol also requires a full 512 bit pseudo random number r. But since r must be stored somewhere in the card while it's squared modulo n, and since it cannot be stored in E^2PROM for the above mentioned reasons, the available amount of RAM only allows to use a 256 bit pseudo random value.

4. Fiat and Shamir allow r to be taken from the range $[0, n)$. Obviously, if r might be 0, all to do for a foreged identification were always to send $x = 0$ in step 3 of the protocol. The implemented pseudo random generator also may produce $r = 0$ with a very small probability, but the PC program prevents a successful identification with $x = 0$.

IV. The Algorithms

In addition to the virtually 'trivial' tasks like communication or managing the protocol itself there are two subroutines in the protocol runtime programs that have to be carefully considered, namely the pseudo random number generator and the modular multiplication.

The pseudo random number generator consists of 12 cascaded cyclic shift registers implemented in software. Gollmann [2] proved that the linear complexity of the sequence generated by cascaded cyclic shift registers grows exponentially with their number. The initial state of some of these registers is derived from the uninitialized RAM immediately after power-on or from the value of a free running on chip timer. The statistical properties and the possibility of physical manipulation of these physical or pseudo-physical random processes are not yet further examined. However, the remaining pseudo random generator should be strong enough to prevent tampering even if they could be made deterministic.

The modular multiplication is done as a full integer multiplication with successive reduction. Owing to the shortage of RAM space, recursive multiplication algorithms like Toom-Karatsuba seem not to be feasible. Thus a bytewise multiplication and addition using the processor's built-in multiplication instruction

is performed. As the architecture of the smart card processor enforces to use this algorithm, the optimization of this arithmetic was a main goal. As a result some self-modifying code was developed, that must be executed in RAM. However this code does not require as much space as the data of a recursive algorithm would.

In a first version of the implementation the reduction was done bitwise. This solution had two major disadvantages. Firstly, considering time, the bitwise reduction dominated over the bytewise multiplication. Secondly, as the lack of RAM prevented the modulus being shifted bitwise during the reduction, it had to be stored eight times, each time shifted by one bit, and so occupied space that could better be used for more signature values s_i. Although the protocol may be repeated several times to increase its security, every repetition has a considerable communication and computation overhead. Thus it is desirable to store as much signature values as possible to gain an acceptable security with only one protocol pass.

The final implementation uses a method to lead back reduction to multiplication published by Mohan and Adiga [5]. Let Q_0 be the value to be reduced modulo n, with

$$Q_k = 2^{512} Z_k + R_k \quad \text{for } k = 0, 1, \ldots \text{ and}$$

$$Z_k, R_k \in [0,2^{512}).$$

Obviously for

$$\begin{aligned} Q_{k+1} &= Q_k - 2^{512} Z_k &&+ 2^{512} Z_k - n Z_k \\ &= Q_k - 2^{512} Z_k &&+ (2^{512} - n) Z_k \\ &= R_k &&+ (2^{512} - n) Z_k \end{aligned}$$

we have

$$Q_k \equiv Q_{k+1} \pmod{n}.$$

Hence all to be done is to multiply the "upper half" of Q_k by $d = 2^{512} - n$ and add the result to the "lower half" of Q_k. This is a rather straightforward extension of the widely known method for performing reductions modulo 2^m-1 (cf. [4] p. 272). Let $\#X$ denote the length of the binary representation of X in bits. We get

$$\#Q_{k+1} \le \#d + \#Z_k \quad \text{if } \#Z_k \ge \#R_k \text{ or } \#d \ge \#R_k \quad \text{and}$$

$$\#Q_{k+1} \le \max(\#d + \#Z_k + 1, \#R_k + 1) \quad \text{if } \#Z_k < \#R_k \text{ and } \#d < \#R_k,$$

what implies that if

$$\#d \le 256$$

can be achieved, then

$$\#Q_2 \leq 513.$$

This means that after two iterations of multiplication and addition there are at most two additions of d to be done to obtain a result reduced to be less than 2^{512}. The complete reduction eventually necessary may be left to the superior computing capabilities of the PC. Due to the simple multiplication algorithm used, the addition of dZ_k to R_k can be combined with the multiplication to have no extra cost in computing time. The greatest advantage of this reduction algorithm is however that only one 256 bit value d instead of eight 512 bit values n have to be stored within the card's scarce memory.

Concerning the precomputation programs, the condition $\#d \leq 256$ leads to the above mentioned condition $2^{512} > n \geq 2^{512} - 2^{256}$. The remaining problem is to find p and q so that n satisfies this interval condition. Mohan and Adiga propose to use a modulus that has not only two large but also some small prime factors. During the implementation of the reduction it turned out that enough prime pairs can be found which satisfy this condition, so that no additional small primes are needed.

Trying to combine two primes out of a precomputed set of large primes could be shown to be impractical. The simple but effective method implemented is to find a suitable prime p, perform a large integer division to compute a factor q so that pq is within the desired range and to test whether q is also prime. In detail:

Given

$$p < 2^{256}, p \text{ prime and chosen at random}$$

then

$$q = \lfloor (2^{512} - 2^{256} - 1) / p \rfloor + 1$$

satisfies

$$2^{512} > pq \geq 2^{512} - 2^{256}$$

The prime number theorem tells us that randomly chosen value p of a magnitude of order 2^{256} is prime with a probability of about $1 / \ln 2^{256} \approx 0.0056$ (cf. [6] p. 64). Chosing p to be less than 2^{256} ensures that at least one multiple kp of p falls into the interval $[2^{512}-2^{256}, 2^{512})$ of length 2^{256}. q is the least such k. All integers within a small interval around q are slightly larger than 2^{256}. Thus the probability for any of them to be prime is slightly less than $1 / \ln 2^{256}$.

The probabilistic Rabin-Miller test ([4] p. 379), is fast enough to find a suitable prime pair within some dozens of hours on a SUN-3.

V. The Implementation

The smart card's part of the scheme is implemented in its processor's assembly language. The complete program including serial communication and programming of the data (s_i, P, d) into E^2PROM, excluding this data itself, consists of less than 700 bytes of code. As the data programming routine is used only once, it is transfered to and executed in RAM and reprograms itself with data. All 256 bytes RAM are needed for data or code storage or as stack.

The personal computer as the smart card's counterpart is programmed in C. Due to its greater performance it can use the same modular multiplication algorithm as the card without effect on overall execution time. The primality testing was done as background job on some SUN-3 computers.

The current implementation allows the authentication of a 120 byte identification string at a security level of 2^{-20} within an average time of about 6 seconds from card initialisation to acceptance of the identification string.

VI. The Conclusions

The goal of specialised processor architecture must be to implement the most time and space consuming tasks in silicon. So a cryptographic protocol processor for asymmetric protocols should include:

- a 512 bit modulus register and at least two 512 bit registers
- instructions for loading and storing these registers and modular arithmetics
- a buffered serial I/O unit, working independently from the CPU
- a physical random number generator or at least a hardware pseudo random number generator
- some general purpose registers and some RAM as return stack
- a reduced general purpose instruction set
- as much E^2PROM as possible

VII. Acknowledgements

I would like to thank Dr. L Schaumüller, W. Schlapak and H. Eilmsteiner (VOEST-ALPINE AG) as well as Prof. Dr. Th. Beth, Dr. M. Clausen, Dr. D. Gollmann and H.-P. Rieß (University of Karlsruhe) for the support, ideas and discussions contributing to this project.

VIII. Bibliography

[1] A. Fiat, A. Shamir: How To Prove Yourself: Practical Solutions to Identification and Signature Problems, Proc. of CRYPTO 86, Springer LNCS 263, pp. 186 - 194, 1987

[2] D. Gollmann: Linear Recursions of Cascaded Sequences, Contrib. to General Algebra 3, Proceedings of the Vienna Conference 1984, Hölder-Pichler-Tempsky, 1985

[3] ISO: Draft International Standard ISO/DIS 7816-3, Identification cards - Integrated circuit(s) cards with contacts - Part 3: Electronic signals and exchange protocols, 1987

[4] D. E. Knuth: The Art of Computer Programming, vol. 2: Seminumerical Algorithms, Addison-Wesley, 2nd ed. 1981

[5] S. B. Mohan, B. S. Adiga: Fast Algorithms for Implementing RSA Public Key Cryptosystem, Electronics Letters Vol. 21 No. 15, p. 761, August 1985

[6] H. Riesel: Prime Numbers and Computer Methods for Factorization, Birkhäuser 1985