Fast Implementation of AES Cryptographic Algorithms in Smart Cards

Chi-Feng Lu ; Yan-Shun Kao, Ph.D.
Division of Research & Development
NexSmart Technology, Inc.
3F-2, No. 23, Sec. 6, Min Chuan E. Road., Taipei, Taiwan 114, ROC

Hsia-Ling Chiang[†], Ph.D.
Department of Electrical Engineering
Kuang Wu Institute of Technology
No. 151, I-te Street, Peitou, Taipei, Taiwan 112, ROC

Chung-Huang Yang[†], Ph.D.
Institute of Info. and Computer Edu.
National Kaohsiung Normal University
No. 116, Ho Ping 1st Road, Kaohsiung, Taiwan 802, ROC

ABSTRACT

The National Institute of Standards and Technology (NIST) of US announced Rijndael algorithm as the advanced encryption standard (AES) on October 2000, despite AES is surpass in security than data encryption standard (DES), it is still rare to be implemented in smart cards, due to the reason of deficient in AES coprocessors. Here a chip operation system (COS) called NexCard, which derived from Microsoft's Windows COS, is used as the AES implement platform. After a suitable COS architecture design for AES and methodology of efficient memory usage, the simulation result shows that direct embedding AES Encryption may attain 0.56ms at system clock 15 MHz on the INFINEON SLE66CX322P chip without existence of coprocessor.

Corresponding to the development needs in smart card cryptographic algorithms implementations, and different level of the security design specifications, a concept to conjoint numbers of algorithms into single smart card called cipher system on demand (CSOD) method is accomplished in this study concurrent. This is a method utilizing the multi-application capability of NexCard v2.0 to execute same AES algorithm as an on card applet. Although the performance of CSOD is not as good as AES embedded method; CSOD can provide same result in the situation of adaptability and extendibility is concerned over performance

Keywords: *smartcard, AES, encryption, virtual machine*

I. INTRODUCTION

Both hardware chip architecture and software chip operation system (COS) [1] security mechanisms consist in smart cards have made them the widely employed security hardware in communication, digital verification and financial applications. Commonly, internal Data Encryption Standard (DES) [2] algorithm computation is offered in most of the smart cards available at current. But regarding the fact that the 56 bits DES algorithm has been broken [3] and while NIST was requesting a new algorithm globally, Rijndael algorithm was chosen and announced to replace the DES as national standard, called Advanced Encryption Standard (AES) [4-5], the smart cards are indeed need to update the existing build-in algorithm accordingly for enhancing security reliability.

According to NIST [6], DES algorithm can be easily broken in few hours with dedicated computer. Assuming that one could build a "DES Cracker" machine that could recover an unknown DES key in one second, then it would take that machine approximately 149 thousand-billion (149 trillion) years to crack a 128-bit AES key. Although, the safety measurements demands the advance crypto algorithm in smart card COS to be implemented, but the fact that, deficient in AES coprocessor is affecting the progress of development. As we know, coprocessor is intending to increase the calculation efficiency in computer process, and the researches indicate the AES at least three times faster than DES. Hence, effectively govern the characteristics of the cautious selection microcontroller; incorporate with an optimized AES algorithm, and efficient memory usage arrangement, an AES smart card chip has proven practical.

Microsoft first set foot in IC Smart Card domain in 1999, and released its first version of WfSC (Windows for Smart Card) [7] base of Windows architecture, later on, a multi-Applications capability included version of WfSC (version 1.1) was officially announced in July 2000. Since the international smart card specific standards were published by populating international organizations, for instance, the GOP (Global Open Platform) specification, it move the essence of Multi-Applications to a new level, and to satisfied the security concerns in the e-society, PKI mechanisms were closely bounded into the core of varies COS. Base on those reasons, Microsoft released the advanced WfSC COS specification (version 2.0), which includes GOP specification and PKI functionalities. With the maturity of WfSC 2.0 specification, this study is selecting WfSC 2.0 as our implementation platform.

WfSC v2.0 was originally developed on Atmel

[†] This research was supported by the NexSmart Technology, Inc., Taipei, Taiwan, R.O.C.

microcontroller a flash memory chip. In this study, we consider the high level of security requirements in hardware established in the international information security standard published by Common Criteria, the study is taken approach with form of masking COS into the microcontroller, Infineon SLE66CX322P [8] is the target microcontroller for our WfSC 2.0 COS implementation and named NexCard 2.0. Since the SLE66CX322P contains an 8051-compatible microprocessor, 136-Kbyte ROM, 32-Kbyte EEPROM, 5K-byte RAM, an 1100-bit arithmetic coprocessor, and two auto-reload timers, which is a powerful device suitable for the implementation of both symmetric and asymmetric cryptographic algorithms on the smart cards. The NexCard 2.0 not only satisfies WfSC 2.0 specification, the built-in cryptographic algorithms also include RSA, DES, DES3, SHA1 etc., which is a full functional PKI card. The infrastructure of NexCard 2.0 is shown as Figure 1.

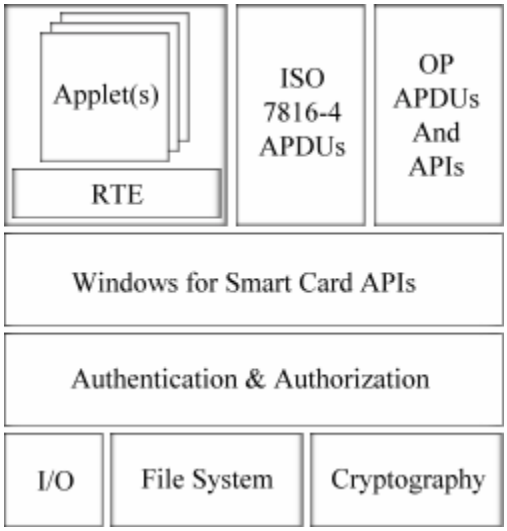


Figure 1. NexCard COS Architecture

The study proposed two approaches to implement the AES algorithm on NexCard 2.0. First approach, the algorithm is embedded into foundation level of COS architecture, the Cryptographic layer, as normal processes. This method is called “Embedded Method”. The second approach is using high level programming languages to interpreted the algorithm in form of an applet, which sits in the top level of the COS architecture, the Applet layer, this approach is called “CSOD Method”. Two approaches will be illustrated and manifested in detail in this paper.

II. AES EMBEDDED METHOD

A COS is integrate with Communication, Command Sets, File System, Authorization, and Cryptography layers [9], cryptographic calculations are only just part of the COS architecture, to enhance the AES calculation, an optimized calculation design, the efficient memory usage and considering the limitation in resources and processing power available in smart card are essential.

2-1. OPTIMIZED AES ALGORITHM

NIST defined AES (Rijndael algorithm) is an iterated block cipher [10] symmetry algorithm. It consists with one fixed size of 128-bit data block and accompanies with one 128, 192, or 256 Bits key block, each data block and key block size is determined independently. The calculation flow [See Appendix A for details] of AES encryption is shown in Figure 2.

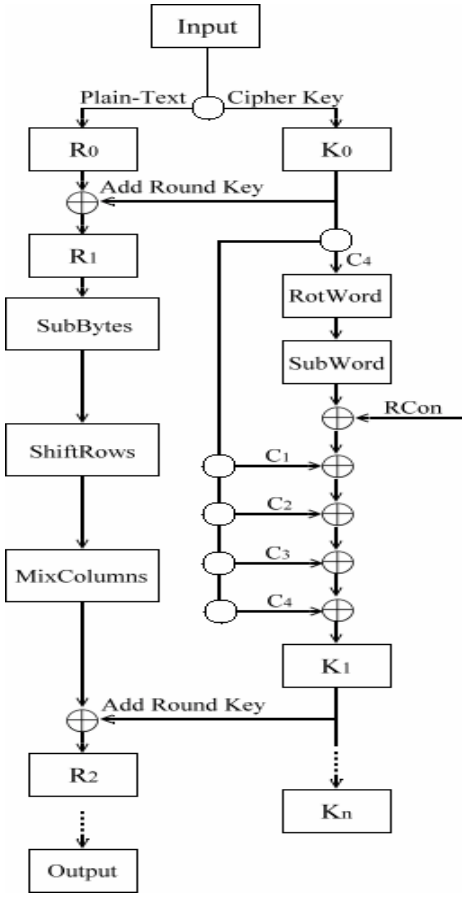


Figure 2. AES Computation Flow

There are 4 basic operations in AES computation flow:

1. AddRoundKey

   Individual element (Byte) in plaintext is updated with the result of XOR calculation of each element (Byte) of round key and element of plaintext

2. SubBytes

It is a non-linear byte substitution, with use of a substitution table (S-Box) which is constructed by composing two transformations, first, polynomial m(x)('11B') is ported, followed by an affine transformation.

3. ShiftRows

   The rows of the plaintext block are cyclically shifted over different offsets.

4. MixColumns

   Transformation of four elements in each column of the plaintext by a polynomial multiplication.

First round of AddRoundKey process is initiated after plaintext and cipher key are entered, every following round of process has to go through SubBytes, ShiftRows, MixColumns and AddRoundKey computations, the number of rounds are depended on the length of cipher key, from 10 to 14 rounds [4]. Note the MixColumns process is not performed in the last round of AES algorithm.

The suggested optimized AES computation flow is as follow:

a) Swap SubByte and ShiftRow

   On considering the best assembly code combinations and continuance memory usage, the order of ByteSub and ShiftRow process in Figure 2 are swapped, to reduce the number of times in memory reads and writes, as well as increase the computation speed without compromised the actual result. The result of the original process [4] (Figure 3(a)) is identical to the one computed with swapped process (Figure 3(b)).

Start of Round				SubBytes				ShiftRows			
19	A0	9A	E9	D4	E0	B8	1E	D4	E0	B8	1E
3D	F4	C6	F8	27	BF	B4	41	BF	B4	41	27
E3	E2	8D	48	11	98	5D	52	5D	52	11	98
BE	2B	2A	08	AE	F1	E5	30	30	AE	F1	E5

(a) Original Order of Process Flow

Start of Round				ShiftRows				SubBytes			
19	A0	9A	E9	19	A0	9A	E9	D4	E0	B8	1E
3D	F4	C6	F8	F4	C6	F8	3D	BF	B4	41	27
E3	E2	8D	48	8D	48	E3	E2	5D	52	11	98
BE	2B	2A	08	08	BE	2B	2A	30	AE	F1	E5

(b) Proposed Order of Process Flow

Figure 3. SubBytes and ShiftRows Swapping

b) Rearrange RoundKey Generation

In the key schedule (key expansion) of the standard AES encryption process, if the calculation order is also rearranged, the number of memory reads and writes can further reduced. For instance, with the standard Key Expansion in 128-bit AES encryption, a 16 bytes key is to be expanded to a 176 bytes round key. To illustrated each byte of the original cipher key and produced round keys are represented in array K. The K_0 to K_{15} are represented as original Cipher Key, value of the key produced in each round can be calculated as formula in Figure 4.

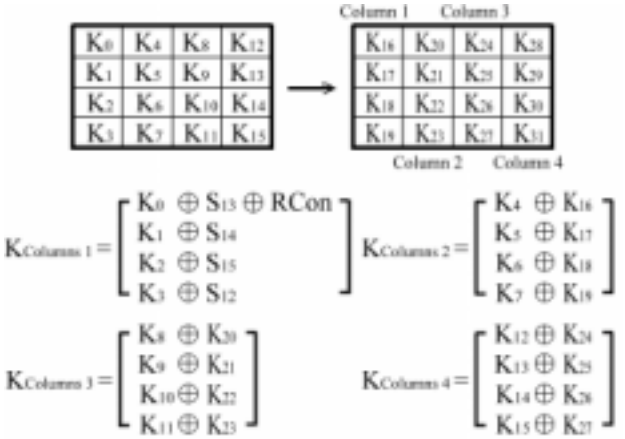


Figure 4. Standard Cipher Key Arrangement.

Let XOR denotes the bit-wise exclusive-OR operation. If the direct addressing (Figure 5) is used in above calculations, the elements of $K_{Row\ 1}$ can be computed by this formula. First, K_0 XOR SBOX [K_{13}] follow by an XOR with RCon to get K_{16}, then sequentially, getting K_{20} by K_4 XOR K_{16}, get K_{24} by K_{20} XOR K_8, and finally get K_{28} by K_{24} XOR K_{12}. The K_{Row2} to K_{Row4} is using the same calculation to computer the cipher key for the next round.

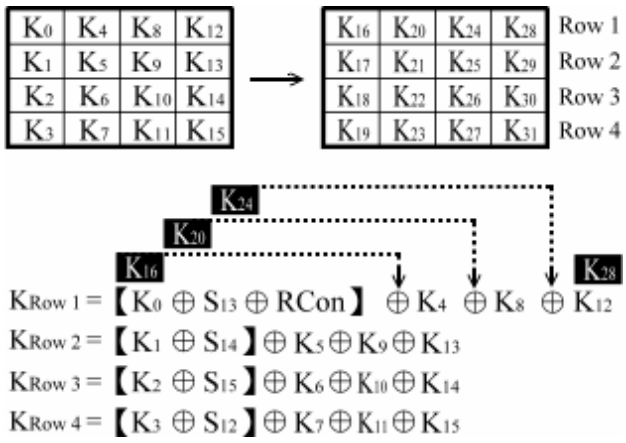


Figure 5. Suggested Round Key Process

>From the proposed key schedule, total only 16 operations of memory accesses have been done, during key expansion from the first round key (K_0 to K_{15}) to next round key (K_{16} to K_{31}), comparing with the standard key arrangement's 32 operations, 50% of memory access operations are reduced. By adopting this suggested rearrangement, the round key in the memory which produced in previous round could be immediately overwritten with the new one. And the required memory space can be cut down to the lowest, which is better fit with the smart card's limitation in memory.

c) Reduce Array Tables

MixColumn process is known to be the major performance bottleneck of AES implementing on the SLE66CX322P, because of the 8051 microcontroller characteristic, which has limited support in 16 bits computation. Due to this difficulty, we replace the square array computation with looking up pre-computed array tables and in conjunction with XOR calculation. Refer to the ANSI C code published in NIST web site, two tables are used in the MixColumn process, However, to prevent from the downgrade in performance caused by the intensive register usage in table look up, the originated table constructing method suggested by NIST was discarded; instead, a modified table constructing method was used as a base on the MixColum calculation characteristic, a single table is produced. By this method, the actual code size is reduced, and results in a better performance.

2-2. COS MEMORY PLANNING

In this study, COS memory planning should not only consider efficiency in AES computation, but also keep in mind in reserving enough memory for other functions of the COS. Two different executions approaches *stand alone calculation* and *on-the-fly calculation* are taken during key expansion and cipher processes to achieve best efficiency and memory usage. Stand alone means the key expansion and cipher processes are calculated separately, on the fly approach is combine key expansion and cipher into one single process.

In AES algorithm, the size of encryption block in each round is fixed (16 bytes). That is, both independent and conjunction approaches will use the same amount of memory during the encryption process. To find the better efficiency approach, we only need to focus on the memory uses during key arrangement.

a) Stand-alone calculations

While using this approach, all round keys are stored in the memory after key arrangement processes are completed, the encryption process desired round keys are remaining in the memory until they are revoked, the keys can be extracted in sequence when needed without re-compute. The memory required for this key arrangement can be calculated using the following formula:

$$R_{KeyLen} = KeyLen + (Nr * KeyLen) \text{------ (1)}$$

where KeyLen is the cipher key length; Nr is the number of rounds. For 128, 192, and 256 bits long key, the memory usages is 176 bytes, 312 bytes, and 480 bytes (see formula (1)) accordingly. However, only XRAM [8] and EEPROM have sufficient memories available in 8051 microcontroller.

Either, IRAM or XRAM can be used for the temporary storage area, the data stored is to be cleared when the reset is instructed to the microcontroller. That is, if the round keys is stored in XRAM, the encryption process has to be executed immediately after the key arrangement, before the memory loses the stored data by reset instruction or used by other COS functions. After consider the meaning of independent key arrangement and encryption, and ensure the smooth operation of other elements and functions in COS architecture, select XRAM as temporary data buffer is not the best choice. If EEPROM is chosen, because of its characteristic, data stored will not be cleared by reset or power off, so, optimizing key arrangement and encryption processes with the independent approach is possible, but one thing has to be aware of is the timing in moving data form EEPROM out to IRAM.

The time needed for moving data out from EEPROM to IRAM is calculated as:

$$t_{pr} = K + V / (f_{sys}) \text{ ---------- (2)}$$

K: Constant part of the execution time.
V: Variable part of the execution time.
f_{sys}: System clock frequency.

$$V = N * p + offset \text{ ---------- (3)}$$

N: Number of programmed bytes in one EEPROM page
p: Multiplication factor
offset: additive offset

According to the official data sheet from INFINEON SLE66CX322P, the parameters K = 4.48, p = 0.06, offset = 0.27, and with System Clock = 15 MHz , the time to move one AES-128 key from EEPROM to IRAM (see formula (2) and (3)) is t_{pr} = 4.562 ms, AES-192 is t_{pr} = 4.594 ms, and AES-256 is t_{pr} = 4.626 ms. If AES encryption is to be performed, the total time needed for AES-128 to move all round keys to RAM is 4.562 * 11 rounds (10 + 1 rounds) = 50.182 ms, AES-192（12 + 1 rounds）4.594 * 13 = 59.722 ms, and AES-256（14 + 1 rounds）4.626 * 15 = 69.39 ms. So, using this independent approach, addition of 50ms to 60 ms is taken in moving round keys out of memory, on top of the time required in key arrangement and cipher process. Comply with the objectives of efficient memory usages and optimized computation performance, using EEPROM to be the data buffer is also inappropriate; hence, this independent approach is not the ideal way to accomplish a high speed AES algorithm implementation.

b) On-the-fly calculations

This approach is to combine the key arrangement and cipher process into one single process; in each round the cipher process is performed right after the key set is constructed, and the current key set is to be replaced by the next key set, and ciphering is done again, cycling the same process until end of the required computation round.

Follow the AES standard computation process, two times of round key length is required for the memory, by calculating different key length AES algorithm. The memory usage for AES-128 is 128 Bit * 2 = 32 Bytes, AES-192 is 48 Bytes and AES-256 is 64 Bytes. If the optimized key arrangement method described in section 2-1 is used, the actual required memory space is equal to the key length of the AES algorithm. That is, the required memory can be reduced to 16 bytes for AES-128, 24 bytes for AES-192, and 32 bytes for AES-256. Obviously, concluding the contentions in point a and b in section 2, the efficiency of key expansion and cipher combination process is better than they are calculated independently.

III. CSOD METHOD

The design aspect of CSOD is to enable incorporation of new international or proprietary algorithm programmed in C or VB languages in COS as the form of applets. To achieve it, a virtual machine (VM) is constructed in COS to interpret the external applets (bytes code) to COS executable instructions.

As the standard AES algorithm procedure, a completed AES applet is consisted with two separate applets, AES Key Expansion and AES Cipher. Due to fact, the applets are all stored in EEPROM before executed, so, no matter the independent or conjunction approach is used, the data moving out from EEPROM step can not be avoided, but using independent approach can cut down the key arrangement process and increasing the efficiency; hence, CSOD method using the independent approach in its process. The procedures are illustrated in Figure 6.

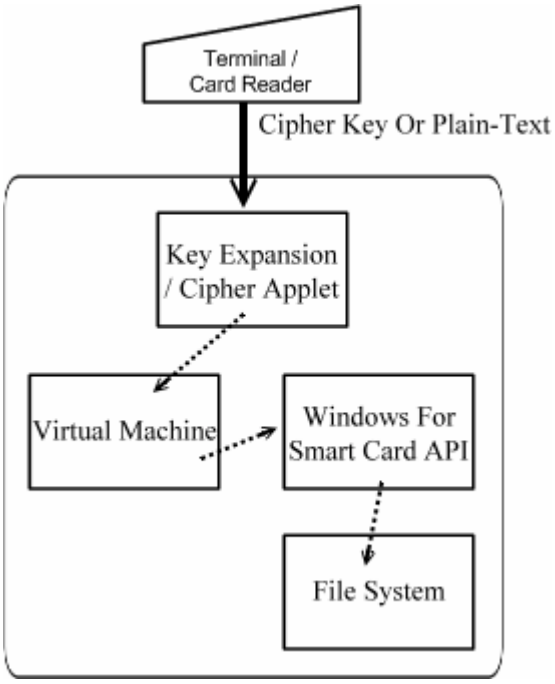


Figure 6. Internal COS Process of Key Expansion and Cipher Applets

After on-card applets (key expansion or cipher applet) received external entry of cipher key or plaintext, the applet byte codes is interpreted by the virtual machine and execution is initiated, during the execution, if COS resource is needed, the on-card API is call to assist the execution. Upon of completion, the calculated round key or ciphered data is saved to File System through the API.

The AES key schedule applet and encryption applet are transferred from the host PC through smart card reader and saved at the EEPROM area of WfSC v2.0, then the applets could be executed by the VM. This procedure,

including transmission and storage then execution, took approximately 1 second for key schedule applet and 5 seconds for encryption applet on the ATMEL AT903232C -based virtual machine.

IV. IMPLEMENTATIONS

A time consuming process, ROM Masking is mandatory for INFINEON SLE66CX322P microcontroller to produce a physical card, all data produced in this paper is collected by execute the program in the KSC SLE66CX322P in-circuit hardware emulator. The COS design and AES algorithm optimization described in this study is base on the released AES algorithm specification from NIST. The efficiency comparison of different AES key length is shown in the following table.

Table 1. Computation Efficiency with Infineon SLE66CX322P Microcontroller

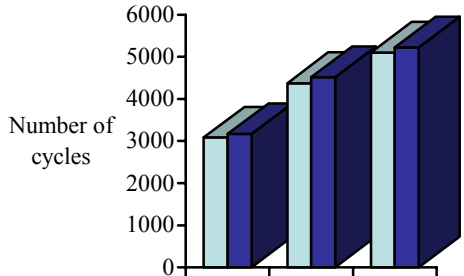
Key length	Number of cycles	Speed	Code length
128	8460	0.56 ms	1013 Bytes
192	12580	0.83 ms	1352 Bytes
256	14875	0.99 ms	1160 Bytes

Note：Internal System Clock : 15 MHz

Unfortunate, there is not available AES on-card under similar conditions to compare with our study, the AES algorithm implementation on Intel 8051 chip by the original AES author is compared instead. The throughput is converted into 8051 chip; the result is illustrated in Figure 7.

In Figure 7, the result clearly shown the final throughput of this study is better than the author publication.

In testing CSOD method, we select ATMEL AT903232C as the target controller; the NexCard COS is flash burned to the microcontroller, before the AES applets is loaded on to it. With the physical card testing, the encryption takes about five seconds, the result can not compete with the AES embedded method, but it is still reasonably acceptable under human operation.



	AES-128	AES-192	AES-256
Our Implementation on Intel 8051	3085	4377	5101
Daemon/Rijmen's Implementation on 8051	3168	4515	5221

Figure 7. The Comparison of Daemon/Rijmen [5] and Ours

V. CONCLUSIONS

AES Embedded Method has proven applicable with the data encryption time around 0.5ms to 0.9ms. Either the microcontroller is co-processor powered or not, the practicability of implementing AES algorithm on smart card has been validated. With the varies and developing cryptographic algorithms in Europe, having a smart card platform capable of processing middle and high level languages with fast turn-around time like CSOD is an advantage. Even though the AES CSOD encryption takes around 5 seconds, this drawback is overcame by the flexibility, convenience and security which CSOD smart card offered.

VI. REFERENCES

[1] Wolfgang Rankl and Wolfgang Effing, Smart Card Handbook, 2nd edition, John Wiley & Sons,2000.

[2] National Institute of Standards and Technology, FIPS PUB 46, “Data Encryption Standard (DES),” January 15,1977.

[3] Matthew Nelson, “Cracking DES code all in a day’s work for security experts,” January 21, 1999. See http://www.cnn.com/TECH/computing/9901/21/descrack.idg/index.html

[4] National Institute of Standards and Technology, “Advanced Encryption Standard (AES),” Federal Information Processing Standard, FIPS PUB 197, November 26, 2001. http://csrc.nist.gov/

publications/fips/fips197.pdf
[5] Joan Daemen and Vincent Rijmen “AES Proposal：Rijndael,” Document Version 2, March 9,1999.
[6] NIST, AES Questions and Answers, http://www.nist.gov/public_affairs/releases/aesq&a.htm
[7] Microsoft Corp., Smart Card for Windows
[8] SLE 66Cxxxp Security Controller Family Data Book, Infineon Technologies AG, September 2002.
[9] ISO 7816 Part 1 to 6: Identification Cards – Integrated Circuit(s) Cards with Contact,1987 to 1996.
[10] J. Daemen and V. Rijmen, “The Block Cipher Rijndael,” *Smart Card Research and Applications, Springer-Verlag LNCS Vol. 1820,* J.-J. Quisquater and B. Schneier, Eds., 2000, pp. 288-296.