Analysis of Advanced Encryption Standards

Minal Moharir

Lecturer, Dept of ISE,
R.V. College of Engg.,
Banglore-59.
minalmoharir@yahoo.com

Dr. A V Suresh

Prof. & Head Dept of IEM,
R.V. College of Engg.,
Banglore-59.
dravsgud@yahoo.co.in

***Abstract*—The Advanced Encryption Standard (AES), the block cipher ratified as a standard by National Institute of Standards and Technology of the United States (NIST), was chosen using a process markedly more open and transparent than its predecessor, the aging Data Encryption Standard (DES).**

Fifteen algorithm were submitted as to NIST in 1998 , NIST choose five finalist.

NIST primary selection criteria are security, performance, and flexibility. This paper enlightens the last two criteria. In this paper we have discussed software performance of five AES finalist.

The paper specifically compares performance of the five AES finalist on a verity of common software platform: 32-bit CPU(both large and smaller microprocessors, smart cards, embedded microprocessors) and high end 64-bits CPUs.

1. INTRODUCTION

Security is not only the most important, but also the most difficult characteristic to compare. In the absence of any theoretical ways of measuring security, we can only fall back on estimates and guesses. "I can't break this algorithm, and all those other smart people can't either" is the best we can say. Hence, all discussions about security rely on this type of non-rigorous argument. When looking at the published cryptanalysis on the AES finalists, it is important to keep in mind what the data mean. Historically, cryptanalytic results against any algorithm have improved over time. Initial results might cryptanalyze a simplified variant of the algorithm, or a version of the algorithm with fewer rounds. Later results improve on those initial results: more rounds or less simplification. Finally, there may be a successful attack against the full algorithm. This is why the published cryptanalysis against the AES finalists, even though none of the results approach practicability and none of the attacks are of any use against the full version of the algorithms, are so important. By comparing how close the published attacks come to breaking the full algorithms, we can get some inkling about how the algorithms' security compares. This is not a perfect comparison by any means, but it is the best we have to go on.

2. SAFETY FACTORS

The best measure of security that we have come across is the *safety factor*. Eli Biham first compared the AES candidates in this manner when he calculated the "minimal secure rounds" [Bih99]. Lars Knudsen also used this factor when he discussed the AES candidates in his first-round comments [Knu99]. Let n be the number of rounds of the full cipher, and b be the largest number of rounds that has been broken. The safety factor . is defined as . s= n/b. A broken cipher has a safety factor of 1. A safety factor of 2 corresponds to a cipher for which a version with half the rounds has been broken. In this context we are very liberal in our definition of what it entails to break a cipher. The most straightforward type of breaking is to find a key-recovery attack: an attack that recovers the key faster than a brute-force search. However, we also include any other non-random property that can be detected faster than an exhaustive search of the key space in our definition of "breaking" a cipher. This can include a statistical test that distinguishes the cipher from a random permutation, detectable relationships between encryptions with different keys, or more generally any detectable property that an ideal cipher would not have. This definition is fairly arbitrary, but it is the best we have found given the situation. Excluding certain types of attacks as "unfair," or certain detectable properties as "unimportant," would be even more arbitrary. Including unsub- stantiated claims of the form "I think I can break x rounds," or "this property might lead to an attack" make the measurement completely arbitrary. It is also reasonable not to consider any other type of simplifications, such as modifying the rounds themselves. It is trivial to attack Rijndael with a linear S-box, or Twofish without the PHT. Taking any attacks of that type into consideration would also lead to completely arbitrary measurements. The biggest inherent problem in our definition of safety factor is that it favours ciphers on which little cryptanalysis has been done. Unfortunately, this is unavoidable if we want to try to maintain at least some kind of objectivity.

There are two problems in applying this definition to the AES finalists. The first one is that most attacks are against the 256-bit-key versions of the algorithm. Thus we might have a reasonable amount of data on how many rounds we

DOI: 10.5176_2010-2283_1.1.24
© 2010 GSTF

can break of each cipher in *2^256* steps, but there is very little information on how many rounds we can break in either 2^128 or 2^192 steps. We therefore use the largest number of rounds broken by any attack on any one of the key sizes. This gives a fairly accurate result for 256-bit key sizes, and introduces a bias for smaller key sizes. At first glance this seems unfair to Rijndael, since we compare the number of rounds attackable under 256-bit keys to the number of rounds in the 128-bit-key cipher. However, this is exactly what we do for all the other algorithms. If we had more information on attacks on 128-bit-key versions we could compute safety factors for each of the key sizes, but we simply don't have that information. Rijndael's reduced number of rounds for smaller keys gives it a speed advantage, but it also reduces the safety factor for those key sizes.

The second problem is MARS. Because of its heterogeneous structure, there are several ways of defining reduced-round versions. It is unfair to only count the 16 core rounds, but it is equally unfair to give all 32 rounds the same weight. We suggest the following: give the core rounds a weight of 1, and the mixing rounds a weight of a. (where *a* is a parameter that we still have to choose). An attack on *c* core rounds and *m* mixing rounds would thus give a safety factor of (16 + 16a)/(*c* + *m*.), and all attacks are measured using this metric. Finally, we choose *a* in such a way as to maximize the resulting safety factor. Thus, the weight *a* is chosen to favor the algorithm as much as possible. This is not ideal, but it is the only reasonable way we have found of getting a number that is somewhat comparable to what we get for the other ciphers.

Table 1.
Safety factors for AES finalists and some increased-round variants.

Algorithm Safety factor	Safety factor
MARS	1.90
RC6	1.11/1.33/1.56
Rijndael	3.56
Serpent	3.56
Twofish	2.67
34-round RC6	2.00
18-round Rijndael	2.00
24-round Rijndael	2.67

Rijndael has an attack on 9 rounds [FKS+00a], and for the three key sizes has a safety factor of 1.11/1.33/1.56 (for the three key sizes, respectively). Serpent has a 9-round attack [KKS00b], for a safety factor of 3.56. Twofish has a 6-round attack [Fer99], for a safety factor of 2.67. The results are tabulated in Table 1 and are not very surprising. RC6 and Rijndael have the smallest safety factors. MARS does better, and Twofish better still. As expected, Serpent has the highest safety factor. Keep in mind that a safety factor of 1 corresponds to a broken cipher. Thus, even moderate advances in cryptanalysis could endanger RC6 and Rijndael. In his first-round comments Lars Knudsen recommended that AES should have a safety factor of at least 2 [Knu99]. We strongly support that notion. The worst thing that could happen to AES is a successful attack a decade from now, even an "academic attack." Not only would this create havoc in many systems, it could also endanger confidential data that was encrypted before the break. Given the very sketchy information we have to go on, we simply cannot afford to gamble on a relatively small safety factor. In our opinion, Twofish and Serpent have good safety factors. MARS is close, but RC6 and Rijndael clearly need more rounds. The table shows that 34-round RC6 and 18 round Rijndael would have a safety factor of 2. To raise the safety factor of Rijndael to the same level as that of Twofish would require 24 rounds of Rijndael. Of course, an increase in the number of rounds results in a corresponding reduction in performance. This will have to be taken into account in any comparison with increased-round versions.

2.1 MARS

It is the most difficult task for us to implement MARS on smart cards or other limited resources. MARS has a complex high level structure such as eight rounds of unkeyed forward mixing, eight rounds of keyed forward transformation, eight rounds of keyed backward transformation, and eight rounds of unkeyed backward mixing. Each of the eight rounds consists of so called type-3 Feistel network. In a type-3 Feistel network, input data is segregated into four words. One of them is taken as a pseudo-random function's input and the output is used to modify three other data words. Since MARS has a block length of 128 bits, each word has 32 bit length. There are three disadvantages of MARS when implemented on a smart card. The first is that it needs 2KB table for S-boxes, but it is not serious. The second is the weakness check of extended key on the key schedule. The last is the rotations with variable shift amount. W e discuss the last two disadvantages here.

It is necessary for MARS to implement complicated "weak" measures on the key schedule[3]. The weak keys for MARS are different from those of DES. In the case of DES, you may disregard the problem of weak key because it only increases some potential threats caused by the weak key properties. However, in the case of MARS, since the weak key check procedure is a part of the algorithm specification, you have to check the weak on the key schedule certainly. Otherwise, you may see a terrible result, such as

© 2010 GSTF

differences in cipher text, although it encrypts the same plain text with common key. As mentioned above, the function of checking the weak on the key schedule is primarily needed. Although implementing weak key check is necessary, it is also true that this introduces another problem for smart card implementation. If we check the weak and regenerate extension keys, there is a risk of applying timing attack. The regeneration of extension keys causes difference in processing time and leaks some information on the key. Further study of coding is necessary to avoid this problem.

Table 3. MARS

	RAM (bytes)			ROM	Time
	Total	Int	Ext	(bytes)	(clock)
Encrypt	60	36	24	3,977	45,588
Schedule	512	512	0	1,491	21,742
Total	512	512	24	5,468	67,330

2.2 RC6

RC6 has various parameters and is defined as RC6-*w/r/b* where *w* means the word length, *r* means the number of rounds, and *b* means the length of key with bytes. We wrote the code with the recommended parameters for AES as RC6-32/20/32. RC6 has a simple structure, but the round function includes various operations such as, addition, subtraction, multiplication, and rotations depending on a variable data. Most parts of RC6 are constructed by arithmetical operation. Therefore, we operate almost all operations on the coprocessor. Furthermore, since the coprocessor can operate up to 1,024 bits for operand, we can execute the pair of rotations with constant shift amount in parallel. An n-bit rotation to two data is written as follows: We duplicate each of data and put them on corresponding CRAM area, then multiply them with 2*n*. As a result, we can improve the performance and reduce the size of code. The coprocessor can execute RC6 data encryption efficiently. RC6 has a simple key schedule but needs much iterations and it is not suitable with on-thefly. The key schedule takes four times as long execution time as encryption. There is an idea to improve the key schedule processing time. A precomputed table improves the speed, but also increases the size of code. It omits the computation of 43 initial values (S[i]) with 32-bit word. The modified code copies S[i]s from precomputed ROM table to RAM area instead of computing S[i]s with constant values. It shall reduce about 4,000 clocks. It needs some extra code or table for precomputed table, thus the size of code increases about 150 bytes.

On the smart cards, RC6 has a moderate encryption speed among the finalists, but its key schedule is slower than Rijindael or Twofish. Note that it has been reported that on the 32-bit processor, RC6's performance is faster thanRijndael and Twofish[5].

Table 4. RC6

	RAM (bytes)			ROM	Time
	Total	Int	Ext	(bytes)	(clock)
Encrypt	124	124	0	489	34,736
Schedule	90	90	0	571	138,851
Total	156	156	0	1,060	173.387

2.3 Rijndael

256-bit key is the fastest for on-the-fly key generation, since we can translate the internal key every two rounds.128-bit key is a little slower than 256-bit key, since we need to make extension keys every round. In the case of 192-bit key, since the key length is not the multiple of the block length, it is not so easy to implement on-the-fly key generation. The xtime is an important subroutine for time constancy.It needs modulus operation with the primitive polynomial. Here is an example of straightforward implementation of the xtime(a) algorithm where the original value is stored in A register.

```
RLA
JR NC, SKIP
AND PRI ; PRI means the primitive polynomial.
SKIP:
... ; end.
```

This is a very dangerous code. Since 'AND PRI' operation is operated only when the carry is '1', an attacker can know whether the value excesses 28 or not in this code. We must avoid such an implementation. Therefore, we use some techniques to avoid differences of processing time and thus prevent cryptanalysis using timing attack. Here is an example of xtime(a) operation with constant time, where a is stored in A register.

© 2010 GSTF

RLA
LD B, A
SBC A, A
AND PRI
XOR B

RLA is a instruction of 1-bit leftward rotation for A register. If RLA is carried out, MSB of A register is set to the carry flag. 'SBC A, A' is an instruction which subtract a value in A register and a carry from A register. It means that if the carry flag is '1' then A register has a value 0xff, otherwise A register has a value 0x00.Next we operate AND instruction with PRI for A register. Then we get PRI or a value 0x00 in A register, and we can operate whether 'XOR PRI' or 'NOP' with the same instructions and processing time. The transformation MixColumn is implemented in an efficient way shown in section 5.1 in [4]. We implement the AddRoundKey and data transfers with the coprocessor.

Table 5. Rijndael

	RAM (bytes)			ROM	Time
	Total	Int	Ext	(bytes)	(clock)
Encrypt	34	32	2	700	25,494
Schedule	32	32	0	280	10,318
Total	66	64	2	980	35,812

2.4 Serpent

There are two kinds of implementation of Serpent: ordinary implementation and bitsliced implementation. Here is the result of an ordinary implementation of Serpent. It is not a bitsliced implementation. It needs a 2,048-byte ROM table on the ordinary implementation. Serpent has various rotational operations. As described in MARS implementation, modular multiplication with coprocessor can be used if they improve the performance. Most of the rotations are, however, more efficient with the Z80 operations than with the coprocessor.1-bit leftward or rightward rotations can be implemented with the Z80 operations, and shifts with multiplies of 8-bit are reorder of bytes. We use the coprocessor operations only for 11-bit rotations, XOR, and memory transfer. Due to the architecture of our coprocessor, it is not suitable to efficiently implement three-operand operation used in Serpent. In [2], Serpent can be implemented using under 80 bytes of RAM with onthe- fly. Our implementation needs twice more RAM, because we write it with coprocessor's operation XOR between halves of CRAM with different offsets. It has more rounds than other finalists do, so its performance is not so good as Rijndael or Twofish. The bitsliced implementation will reduce the size of code and required RAM with a little degradation in speed.

2.5 Twofish

In the case that the length of key is less than 256-bit, we need to pad out the original key until it becomes 256-bit.W e implement Twofish with 128-bit key to take the processing time for padding into account. It includes code for padding, and it is a little slower than 256-bit key. There are two models for implementing Twofish, such as Feistel model and non Feistel model[10].W e implement it with non Feistel model. We assume that it is faster than Feistel. We use coprocessor's operations for additions with subkeys, XOR, and memory transfers on CRAM area, but rotations are implemented with Z80's rotations. The performance of Twofish depends on the size of precomputed tables' [10]. We consider that the case of using some tables amounted to 1,536 bytes. This code is compact for processing the key schedule with precomputed tables. It seems be compatible with 2200 bytes for code and table size model in [11].The size of precomputed tables is belongs to encryption code in table 7. Twofish is as fast as DES on throughput. It does not have any exceptional advantages, but we have nothing to complain about the performance.

Table 6. Serpent

	RAM (bytes)			ROM	Time
	Total	Int	Ext	(bytes)	(clock)
Encrypt	68	68	0	3,524	71,924
Schedule	96	96	0	413	147,972
Total	164	164	0	3,937	219,896

Table 7. Twofish

	RAM (bytes)			ROM	Time
	Total	Int	Ext	(bytes)	(clock)
Encrypt	34	32	22	493	31,877
Schedule	56	32	24	315	28,51 2
Total	90	64	26	808	60,389

© 2010 GSTF

3. CONCLUSION

We conclude the performance and the required resources on our implementations in tables. We consider that Rijndael is excellent on all aspects C6 is as good as Rijndael on the code point of view, but the key schedule consumes more time. Twofish needs much ROM memory than RC6 and Rijndael because of the table. It is faster than Triple DES and equal to DES on the throughput.It will have good performance on any smart cards. MARS has disadvantages of its code size caused by four of eight round iterations and a 2,048-byte table. The speed is equal to Twofish's one. We consider MARS has some difficulties to check 'weak' on the key schedule and regenerate. Serpent has disadvantages of its performance caused by the iterations of rounds and the difficulty of key schedule. The bitsliced implementation will improve the requirement of ROM or RAM, but slower than others. We tried to write all program codes to consume as little RAM area as possible.

REFERENCES

[1] R. Anderson, E. Biham, and L. Knudsen, \Serpent: A Proposal for the Advanced Encryption Standard", AES submission, 1998.

[2] R. Anderson, E. Biham, and L. Knudsen, \Serpent and Smartcards", CARDIS '98, 1999, available on http://www.cl.cam.ac.uk/~rja14/serpent.html.

3] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S. M. Matyas Jr., L. O'Connor, M. Peyravian, \MARS -a candidate cipher for AES", AES submission, 1998.

[4] J.Daemen, V.Rijmen, \AES Proposal: Rijndael", AES submission, 1998.

[5] B. Gladman, "AES Algorithm Efficiency", http://www.btinternet.com/~brian.gladman/cryptography technology/Aes/

[6] G. Hachez, F. Koeune, and J. Quisquater, \cAESar results: Implementation of Four AES Finalists on Two Smart Cards", The second AES conference, 1999, available on http://www.dice.ucl.ac.be/crypto/CAESAR/caesar.html.

[7] H. Handschuh, and P. Paillier, \Smart Card Crypto-Coprocessors for Public-Key Cryptography", CryptoBytes, Vol. 4, No. 1, RSALaboratories, 1998.

[8] G. Keating, \Performance Analysis of AES candidates on the 6805 CPU core", The second AES conference, 1999, available on http://www.ozemail.com.au/~geoffk/aes-6805/.

[9] M. Matsui, \New Block Encryption Algorithm MISTY", Fast Software Encryption, 4th InternationalWorkshop Proceeding, LNCS 1267, Springer-Verlag, 1997, pp.54-68.

[10] National Bureau of Standards, \Data Encryption Standard", U.S.Department of Commerce, FIPS 46-3, October 1999.

[11] J. Nechvatal, E. Barker, D. Dodson, M. Dworkin, J. Foti, and E. Roback, \Status Report on the First Round of the Development of the Advanced Encryption Standard", http://csrc.nist. gov/encryption/aes/round1/r1report.pdf

Minal Moharir is working as a Lecturer in R V College of Engineering, Bangalore, India. She received her M.Tech in Computer Network & Engineering from VTU, Belgaum, India. She is currently pursuing her Ph.D at Avinashi Lingam University, Coimbatore, India. She has 8 years of experience in teaching. She has presented many papers in various national & International conferences and reputed Journals.

Dr A V Suresh is working as the Head of the Department in R V College of Engineering, Bangalore, India. He has completed his Ph.D from Mysore University in 2003. He has 21 years of experience in teaching and 8 years of research experience. His areas of interest is Machine Tool Engineering , Quality assurance, and Operation Research. He has presented many papers in various national & International conferences and reputed Journals.

© 2010 GSTF