Date: Fri, 12 May 2000 11:46:06 +0200 (CEST)
From: Vincent Rijmen <Vincent.Rijmen@esat.kuleuven.ac.be>
To: AESround2@nist.gov
cc: Jim Foti <jfoti@nist.gov>, Daemen Joan <daemen.j@protonworld.com>
Subject: comments

Please find in attachment some comments from Joan Daemen and myself.

Best regards,

Vincent

Vincent Rijmen |
| Pray to God, but keep rowing to shore.
phone +32 16 32 18 62 | -- Russian Proverb
vincent.rijmen@esat.kuleuven.ac.be | on provable security

The AES second round Comments of the Rijndael

Joan Daemen Vincent Rijmen

daemen.j@protonworld.com
vincent.rijmen@esat.kuleuven.ac.be

May 12, 2000

1 Introduction

In this note we firstly give our reaction to a number of publications related to Rijndael, that were presented at the AES and FSE conference. We want to place a number of statements in the right perspective.

Secondly, we provide some supplementary information to the official documentation. At the time of submission, we did not anticipate all the questions and issues that would be brought up. We hope that this note helps to enhance the understanding of Rijndael.

2 Security

2.1 Cryptanalytic attacks

In the original submission [6], we described a theoretical attack, that can break a 6-round version of Rijndael, using 2^{32} chosen plaintexts, and requiring an (off-line) effort of about 2^{72} encryption. For 256-bit keys, this attack can 'trivially' be extended by guessing an extra full round key. This results in an effort of about 2^{200} encryptions. This attack has been improved upon by four groups of cryptanalysts [10, 2, 15, 12].

The best attack for 128-bit keys seems to be given in [12], were an attack is presented on 7 rounds, 'marginally faster than exhaustive key search'. The cost of this extension by a single round is an increase of the workload by factor of 2^{56}, compared to the original 6-round attack. If we compare to the improved 6-round attack published in [10], the increase in time complexity due to adding a single round becomes 2^{88}.

For 256-bit keys, there are theoretical attacks for up to 8 rounds of Rijndael. The cost of this additional round is a factor of 2^{76} in workload and 2^{96} in number of chosen plaintexts required. Moreover, the number of rounds specified for 256-bit keys is 14, still giving 6 rounds of margin.

We conclude that the number of rounds in Rijndael provides a sufficient margin of security with respect to cryptanalytic attacks.

The key schedule In the original documentation [6, p. 28], we stated that one of the design requirements of the key expansion is that 'Knowledge of a part of the Cipher Key or Round Key bits shall not allow to calculate many other round key bits.'

In [10] it was shown that for the case of a 256-bit key, the knowledge of 7 well-chosen round key bytes allows the calculation of 28 bytes of the expanded key. This fact is used as an argument to show that the design requirement is not fulfilled.

However, the primary requirements of the Rijndael key schedule are that it is efficient and can be executed in a small amount of RAM. These requirements follow from our concern for the performance in applications with a high key agility. This limits the choice to relatively simple and invertible operation. Within the range of candidate operations, the key expansion actually chosen was the best we could find with respect to the criterion stated above.

The goal of the Rijndael key schedule is the protection against related-key attacks, attacks based on symmetry and (cryptanalytic) key splitting attacks. In our opinion, the protection against attacks that first obtain key bits via side channel analysis (e.g., DPA) and then reconstruct the rest of the key by cryptanalysis should not be addressed by the key schedule itself. However, we do think that the operations used in the key schedule should be chosen to allow for implementations that can provide protection against side channel attacks.

This 'weakness' in the Rijndael key schedule does not result in a better cryptanalytic attack on Rijndael with 128-bit keys. The best current attack that exploits the *weakness* of the Rijndael key schedule, is the related key chosen plaintext attack published in [10], that breaks 9 rounds of Rijndael with 256-bit keys. The attack is an extension of the Square attack and takes 2^{85} related-key chosen plaintexts and has a time complexity of 2^{224}. It can be considered as an extension of the 7-round attack by two more rounds at the cost of a factor 2^{48} in amount of required chosen plaintext material and simultaneously a factor 2^{50} in time complexity. The fact that 5 more rounds remain to be tackled before a valid (academic) attack against Rijndael can be claimed makes us quite confident in the current key schedule.

2.2 Implementation attacks

Implementation attacks are an important threat in many applications. Remarks upon the candidates' resistance against implementation attacks have been made by several teams, each of them related in a larger or lesser degree to one of the design teams [3, 4, 16, 7].

Since implementation attacks have been described only quite recently, there is not yet a consensus on the best approach to make algorithms and implementations resistant against these attacks. Some authors argue that key-related operations should be complex, in order to make it difficult for attackers to determine key bits from obtained information [3]. Our view is that simple operations should be used, because they are easier to protect by techniques like k-way split implementation [5], and load balancing [7].

2.3 Future resiliency

Some authors place much emphasis on the topic of future resiliency [13]. Of course, it is wise to build in some kind of 'margin' against evolutions in cryptography. On the other hand, since is very difficult to predict the future, it is also very difficult to build in security against attacks that yet have to be developed. Ranking the different algorithms with respect to their assumed resistance against attacks that we cannot imagine now, is therefore very difficult.

2.4 Other results

Pseudo-Randomness In [17], some of the candidates are compared with respect to the number of rounds they would need in order to get a certain level of pseudo-randomness. To

this end, not the actual algorithms are studied, but some of the building blocks are replaced by idealised random functions. The paper than continues by comparing how well suited the different algorithms' structures are in generating pseudo-random outputs. While this analysis can give interesting insights, there is a big *IF* attached. Looking at the results of the analysis, it becomes clear that the size of the idealised building blocks greatly affects the outcome of the analysis. The theoretical minimum number of rounds for pseudo randomness in the case of a Feistel cipher with a 64-bit idealised block is 9; in the case of MARS and RC6, with 32-bit idealised blocks, this number becomes 25; in the case of Rijndael, with 8-bit idealised blocks, this number becomes 384. Serpent, formerly assumed to be the most secure candidate in the whole AES process, gets 4-bit idealised building blocks and is therefore punished with an enormous minimum number of rounds.

The problem with this kind of analysis can best be illustrated with the case of Twofish. Twofish can be considered as a generalised Feistel cipher, and if its 64-bit F-function is idealised, it seems very secure. However, zooming in on the round function, we see that it actually never operates on 64-bit quantities, and we might decide to equip it with idealised 32-bit building blocks. The result will be that the security of Twofish becomes comparable with the security of MARS and RC6. However, zooming in even further on the Twofish design, we see that its nonlinearity is based on 8-bit S-boxes, that are built by iterating 4-bit S-boxes. If we decide to idealise only the 4-bit S-boxes, the theoretical minimum number of rounds raizes to a level well above the Rijndael figure. There seems to be no unambiguous way to decide on the size of the idealised components.

It can be useful to compare the results of [17] with the results obtained in [23, 24], where it is claimed that, in general, SPN-networks, like Rijndael, perform better that Feistel structures.

3 Performance

3.1 Parallelism

The inherent parallelism of Rijndael allows to make optimal use of parallelism on many modern processors [27] and in hardware. Some authors (e.g., [9, 25, 26]) compare performance in applications with interleaving of multiple instances of the cipher. Interleaving allows ciphers with little parallelism to use a parallel processor to its full extent. However, interleaving is not possible in CBC mode, CFB mode or OFB mode, the modes that are currently used for data encryption.

NIST can consider to define new modes of operation, in order to reduce the performance penalty for using a cipher with suboptimal performance on parallel platforms. However, these so-called interleaved modes are sub-optimal in reaching the goals for which the modes were originally introduced. For instance, one of the most important applications of block ciphers today is MACing. The block cipher is applied in CBC mode and (part of) the final block of the cryptogram is used as MAC. The fact that the final block of the cryptogram depends on all blocks of the message is an essential feature. If an interleaved CBC mode would be used, the final cryptogram block only depends on a fraction $1/n$ of the message, where n is the interleaving interval.

3.2 Decryption Performance

In the original Rijndael submission, we explained that in some implementations, the inverse operation of Rijndael may be slower than the forward operation. However, the performance drop should not be overrated. We give a quick overview of the different issues.

Two implementation models Generally speaking, we can distinguish the software implementations of Rijndael in two categories. The first type of implementation combines the operations ByteSub and MixColumn in large tables. This implementation is suited for 32-bit and higher platforms. The implementations of the cipher and the inverse cipher have the same performance. The round keys for inverse operation are derived from the round keys for forward operation by means of a simple procedure.

The second type of implementation implements each operation explicitly, using a few small tables only. The inverse operation is not as performant as the forward operation, but both operations are quite fast compared to the other candidates (cf. [14]). In this type of implementation, the round keys for inverse operation are equal to the round keys for forward operation, but used in a different order. We see that the currently available implementations require a longer key setup for the inverse operation. We explain below how the inverse round key setup can be sped up for most applications.

The key setup For simplicity, we discuss only the case of 128-bit keys, but a similar reasoning holds for the other key lengths.

The key expansion takes the key k as input and applies repeatedly a nonlinear transformation to it. Let this transformation be denoted by r. The round keys k_i are simply the outputs of this transformation r.

$$\begin{aligned} k_0 &= k \\ k_1 &= r(k) \\ k_2 &= r(k_1) = r(r(k)) = (r \circ r)(k) \\ &\dots \end{aligned}$$

In applications where only a few data blocks need to be encrypted, RAM is scarce, or key agility is important, the round keys are usually not calculated in advance, but r is executed in parallel with the actual cipher operations. This technique seems to be used in most implementations on this kind of platforms (cf. [14, 19]).

Most implementations of the *inverse* operation are suboptimal in this respect, because they first go through a setup phase to determine all round keys from the supplied key, before executing the inverse operation, round by round, using the precalculated round keys in inverse order.

We want to point out that for many applications, it might be better to calculate k_{10} only once, and store it for future use. In some applications, where a given machine needs to implement the decryption operation only, the value of the key k can be overwritten with k_{10}. Indeed, because the transformation r is invertible, all round keys can also be derived as follows:

$$\begin{aligned} k_9 &= r^{-1}(k_{10}) \\ k_8 &= (r^{-1} \circ r^{-1})(k_{10}) = r^{-1}(r^{-1}(k_{10})) \\ &\dots \end{aligned}$$

In this way, the round keys are calculated in the order they are needed during the inverse operation.

3.3 Jave performance

Both in the NIST tests [8] and the independently developed implementations of [22], it can be seen that Rijndael performs reasonably well. The results of [11] are based on the performance of the KAT and MCT tests, instead of the performance of the algorithms themselves. Given the difference in performance between Rijndael and Crypton, two very similar algorithms, on these tests, it seems that the influence of the implementation of the tests weighs too heavily on the results.

Perhaps the most striking conclusion can be made by comparing the performance of the 10-round and the 14-round versions of the Rijndael implementations in [8, 22]. It turns out that 80% [8], respectively 27% [22] of the time is spent in operations that do not scale with the number of rounds. It is not clear to us whether this is a Java feature, or something specific for Rijndael. (This could be investigated by measuring the performance of the other candidates for a different number of rounds.)

As a final note, we want to comment on the 'implementation difficulty ratings' in [22]. We agree that implementation difficulty can be a topic to consider. However, in that case, a distinction should be made between the difficulty of making a correct implementation, and the difficulty of getting a performance-optimised implementation.

3.4 Corrections to [20]

The authors of [20] try to give an overview of all the candidates' performance on many different platforms. Given the wide scope of the paper, it seems inevitable that some inaccurracies remain undetected for a long while. We point out some mistakes that in our view are important for the case of Rijndael.

A first inaccuracy is related to the performance of the algorithms on memory-limited 8-bit smart cards. We repeat the authors' figures in the left hand side of Table 1.

Table 1: Smart Card RAM requirements (bytes).

	according to [20]	corrected numbers	
Algorithm	128-bit key	128-bit key	256-bit key
Rijndael	52	36	52
Serpent	50	80	80
Twofish	64	64	88

As explained in the Rijndael submission [6], the 52 bytes are only required for a key length of 256 bits. For a 128-bit key, this number is reduced to 36. Secondly, it is unclear where the authors obtained the quoted number of 50 bytes for a Serpent implementation. The Serpent designers state "less than 80 bytes" [1]. A quick evaluation of the description learns that any Serpent implementation requires at the very least 32 bytes for the key (which is always expanded to 256 bits), 16 bytes for the current round key and 16 bytes for the text. Thirdly, we learn from the Twofish documentation [21], that 64 bytes are required for 128-bit key

lengths. For 256-bit key lengths, at least 88 bytes are required. Another overview of the candidates' performance on smart cards can be found on the web page of G. Keating [14].

A second point that needs some clarification, lies in the various comparisons of assembly implementation of the algorithms. As the authors state clearly, they used the best performance estimates available to rank the candidates. It should be noted here that only Twofish has been implemented with in-compiled key, because many programmers feel that in practice, in-compiled key code will not be used. Leaving this issue to the specialists, it is clear that for the sake of comparison either all algorithms should be implemented in this way, or none at all. While the performance of most algorithms will not change dramatically when in-compiled keys are used, the change will probably be sufficient to disrupt the ranking of the top algorithms as presented in [20].

References

[1] R. Anderson, E. Biham, L.R. Knudsen, Serpent and smartcards, presented at CARDIS '98, available from `http//www.cl.cam.ac.uk/ rja14/serpent.html.`

[2] E. Biham, N. Keller, Cryptanalysis of reduced variants of Rijndael, AES3.

[3] E. Biham, A. Shamir, Power analysis of the key scheduling of the AES candidates, AES2.

[4] S. Chari, C. Jutla, J. Rao, P. Rohatgi, A cautionary note regarding evaluation of AES candidates on smart cards, AES2.

[5] S. Chari, C. Jutla, J. Rao, P. Rohatgi, Towards Sound Approaches to Counteract Power-Analysis Attacks, Crypto'99, LNCS 1666.

[6] J. Daemen, V. Rijmen, AES Proposal: Rijndael, official documentation.

[7] J. Daemen, V. Rijmen, Resistance against implementation attacks: a comparative study of the AES proposals, AES2.

[8] J. Dray, NIST performance analysis of the final round Java AES candidates, AES3.

[9] A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar, An FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalists, AES3.

[10] N. Ferguson, J. Kelsey, B. Schneier, M. Stay, D. Wagner, D. Whiting, Improved cryptanalysis of Rijndael, FSE2000.

[11] A. Folmsbee, AES Java technology comparisons, AES2.

[12] H. Gilbert, M. Minier, A collision attack on 7 rounds of Rijndael, AES3.

[13] D. Johnson, AES and future resiliency: more thoughts and questions, AES3.

[14] G. Keating, `http://www.ozemail.com.au/~geoffk/aes-6805/.`

[15] S. Lucks, Attacking 7 rounds of Rijndael under 192-bit and 256-bit keys, AES3.

[16] T. Messerges, Securing the AES finalists against power analysis attacks, FSE2000.

[17] S. Moriai, S. Vaudenay, Comparison of randomness provided by several schemes for block ciphers, AES3 submission.

[18] D. Patel, The AES winner, AES3 submission.

[19] F. Sano, M. Koike, S. Kawamura, M. Shiba, Performance evaluation of AES finalists on the high-end smart card, AES3.

[20] B. Schneier, D. Whiting, A performance comparison of the five AES finalists, AES3.

[21] B. Schneier et al., Twofish - a block encryption algorithm, AES1.

[22] A. Sterbenz, P. Lipp, Performance of the AES candidate algorithms in Java, AES3.

[23] M. Sugita, K. Kobara, H. Imai, Pseudorandomness and maximum average of differential probability of block ciphers with SPN-structures like E2, AES2.

[24] M. Sugita, K. Kobara, K. Uehara, S. Kubota, H. Imai, Relationships among differential, truncated differential, impossible differential cryptanalyses against word-oriented block ciphers like Rijndael, E2, AES3.

[25] R. Weiss, N. Binkert, A comparison of AES candidates on the Alpha 21264.

[26] T.J. Wollinger, M. Wang, J. Guajardo, C. Paar, How well are high-end DSPs suited for the AES algorithms, AES3.

[27] J. Worley, B. Worley, T. Christian, C. Worley, AES finalists on PA-RISC and IA-64: implementation & performance, AES3.