Date: Wed, 24 May 2000 17:30:17 +0100 (BST)
From: Sean Murphy <sean@dcs.rhbnc.ac.uk>
X-Sender: sean@platon.cs.rhbnc.ac.uk
To: AESround2@nist.gov
Subject: NESSIE comments on AES

We wish to submit the attached postscript file as an AES Round 2
comment. It contains comments from the NESSIE project about the AES.

   Sean Murphy

Comments by the NESSIE Project
on the AES Finalists

B. Preneel[1], A. Bosselaers[1], V. Rijmen[1], B. Van Rompay[1]
L. Granboulan[2], J. Stern[2],
S. Murphy[3],
M. Dichtl[4], P. Serf[4]
E. Biham[5], O. Dunkelman[5], V. Furman[5]
F. Koeune[6], G. Piret[6], J-J. Quisquater[6],
L. Knudsen[7], H. Raddum[7].

24 May 2000

[1] K.U. Leuven, Dept. Elekrotechniek, Kardinaal Mercierlaan 94, B-3001 Heverlee, Belgium.
[2] École Normale Supérieure, Département d'Informatique, 45 Rue d'Ulm, Paris 75230, France.
[3] Royal Holloway, Information Security Group, Egham, Surrey TW20 0EX, UK.
[4] Siemens AG, Otto-Hahn-Ring 6, München 81732, Germany.
[5] Technion, Computer Science Dept., Haifa 32000, Israel.
[6] U.C. Louvain, Département ELEC, PO Box 3, Place du Levant, B-1348 Louvain-la-Neuve, Belgium
[7] U. Bergen, Dept. of Informatics, PO Box 7800 Thormoehlensgt. 55, Bergen 5020, Norway.

Contents

1 Introduction

The NESSIE (New European Schemes for Signature, Integrity and Encryption) project is funded by the European Union's fifth framework programme to assess, via an open call, cryptographic primitives for possible future standardisation. The partners in the programme are Katholieke Universiteit Leuven (Belgium), École Normale Supérieure (France), Fondazione Ugo Bordoni (Italy), Royal Holloway (UK), Siemens AG (Germany), Technion (Israel), Université Catholique de Louvain (Belgium) and Universitetet i Bergen (Norway). Further details concerning the NESSIE project can be found on the NESSIE website http://www.cryptonessie.org.

The document contains comments about the AES process and AES finalists that represent the consensus view of the NESSIE partners.

2 General Comments on Security Evaluation

2.1 Extent of Security Evaluation

We believe that the effort spent on evaluating the security of the five AES finalists has been very limited, certainly compared to the 17 man-years spent by IBM on DES in the 1970s. It is noteworthy that the majority of papers at AES3 dealt with performance evaluation rather than with security evaluation.

There are several reasons for this:

- the tight timing constraints;

- some of the finalists are rather complex, which makes them hard and time-consuming to approach.

Nevertheless we believe that it is unlikely that a practical attack (e.g., using 2^{32} known plaintexts and requiring 2^{70} encryptions) will be found for any of the five finalists within a short timeframe (less than five years). However, it is likely that certificational attacks against some of the finalists will be found, whose complexities will be lower than the size of the key space.

It is certainly true, though, that more cryptanalysis would have been very useful to increase our confidence in the long term security of these algorithms. Moreover, it might have resulted in a meaningful comparison from a security point of view.

2.2 Comparison of Algorithms with Different Security Margins

We agree that it is not easy to define the 'security margin' of a cryptographic algorithm. One of the reasons is that it is a dynamic concept, that is, additional analysis will typically lead to improved attacks and thus decreased security margins.

However, we believe that it is essential to try to define such a margin to compare the security/performance trade-off for the AES finalists.

We are aware that NIST has solicited inputs on this, but it seems that for the time being, no consensus has been reached on the relative or absolute security margins. Some of the reasons why this is so difficult are discussed below.

2.3 Security Margins and Known Attacks

The best known attacks on a block cipher might not reflect the true security level of the candidate nor the state-of-art of cryptanalysis. There are several reasons for this.

1. Transparent designs invite straight-forward attacks on reduced-round versions.

2. The amount of time invested in the analysis of the individual candidates may vary.

3. The experience and capability of the individual cryptanalysts may vary.

Therefore we feel that one has to be careful when comparing the candidates with respect to security margins and known attacks.

2.4 Statistical Methodology

Certain aspects of the statistical methodology employed by NIST (see Soto) are questionable in that they give statistical test results that are misleading or meaningless (see Murphy). We give some examples below.

- The use of a data category that gives no information about the block cipher.

- The use of stream cipher tests to test block ciphers via the arbitrary concatenation of data blocks.

- The highly inaccurate calculation of probabilities for certain types of error.

Statistical Methodology References

S. Murphy, *The Power of NIST's Statistical Testing of AES Candidates*, submitted as comment on AES to NIST, 2000.
J. Soto, *Randomness Testing of the AES Candidate Algorithms*, NIST document, 1999.
N.B. Both the above documents are available on the NIST website.

2.5   Examination of the Diffusion Properties of the AES Finalists

In order to check the diffusion properties of reduced round number versions of the AES finalists, NIST used stream cipher tests. The NESSIE project used well known cryptologic measures for diffusion properties. The degrees of completeness, of avalanche effect and of strict avalanche criterion were determined for full round and reduced round versions of the AES finalists.

As expected, the results for all AES finalists were good. With respect to the quantities measured, they are indistinguishable from random permutations after a very small numbers of rounds. The detailed results are given in the appendix. It should be noted, that these results cannot be interpreted to indicate cryptographic strength or to compare the AES finalists. Therefore the usefulness of these results is limited. However, bad results would have been a strong indication of a problem.

3   Security Evaluation of Selected AES Finalists

The institutions within the NESSIE project have (direct and indirect) connections with three of the AES finalists, so we do not feel it appropriate to make a statement concerning their security evaluation in this jointly–written document. The individuals involved have made detailed personal statements concerning the security evaluations of their own submissions. We thus restrict our comments about security evaluation to the remaining two AES finalists: Twofish and MARS.

3.1   Twofish

In this Section, we summarise some of the comments by NESSIE project members to NIST that relate to the security of Twofish.

An Observation on the Key Schedule of Twofish and *The Key Separation of Twofish* are two papers concerned with a key separation property of Twofish. This key separation property is a division of the key into two separate parts that have completely different

functionality. One part is used in S-Box generation (via XOR) and the other part is used in additive subkey generation. This division of the key into two different functional parts is not a property shared with most other block ciphers, contrary to the claim in *Twofish Technical Report No. 7*. The first paper also demonstrates that the key schedule of Twofish with fixed S-Boxes has some properties that are non-standard for block ciphers with fixed S-boxes. The second paper shows why the argument given in *Twofish Technical Report No. 4* as to the irrelevance of key separation is erroneous.

In *Trawling Twofish (revisited)* statistical properties of Twofish are found. It is shown that such statistical properties may have probability 2^{-256} to get 32 bits of nontrivial information in each of the 16 rounds. This means that for any fixed key, one can expect to find one pair of plaintexts which has the differences specified in the property in each round. For a similar cipher with truly random round functions such properties would have a probability of 2^{-512}. One possible application of these properties is in attacks distinguishing Twofish with a reduced number of rounds from a randomly chosen permutation. It has been shown that such attacks are possible for scaled-down versions of Twofish reduced to four rounds, contrary to the claims of the designers. It is unknown how such attacks will apply for real Twofish and for how many rounds.

In *Differential Cryptanalysis, Key-dependent S-Boxes, and Twofish*, some differentials are derived for a proportion of the keys by exploiting Twofish's variable S-Boxes. The approach is to match the S-Box to the differential, rather than the differential to the S-Box, which is the approach forced by fixed S-Boxes. The variable S-Boxes allow us to find a number of them (corresponding to a fraction of keys) for which the required differential holds with a reasonable probability. A preliminary analysis using this technique suggests the existence of an 8-round attack on Twofish.

A related paper *Differential Probabilities for Twofish S-Boxes* gives a theoretical derivation of the distribution of XOR table that involve the simultaneous use of different differentials over the same S-Box. The paper also tabulates these distributions. This gives a tool for the calculation of the proportions of keys giving a particular differential with a given probability, a technique used in the previous paper.

Cryptanalysis of Twofish is a thesis that explores various properties of Twofish. Its most important observation concerns the derivation of a 5-round differential given in the original Twofish paper. The number of plaintexts needed for this differential is reduced by 2^5.

Furthermore, the key-dependent S boxes of Twofish are claimed to improve the security of the cipher. However, the literature contains several examples where the replacement of fixed S boxes with key-dependent S boxes reduce the strength of block ciphers (see, for example, Biham and Biryukov).

Twofish References

E. Biham and A. Biryukov, *How to strengthen DES using existing hardware*, Asiacrypt 1994.

K. Daemen, *Cryptanalysis of Twofish*, Thesis, KU Leuven, 2000.

J. Kelsey, *Twofish Technical Report No. 7*, submitted to NIST as AES comment, 2000.

L.R. Knudsen, *Trawling Twofish (revisited)*, submitted to NIST as AES comment, 2000.

F. Mirza and S. Murphy, *An Observation on the Key Schedule of Twofish*, 2nd AES Conference, Rome 1999.

S. Murphy, *The Key Separation of Twofish*, submitted to NIST as AES comment, 2000.

S. Murphy, *Differential Distributions for Twofish S-Boxes*, submitted to NIST as AES comment, 2000.

S. Murphy and M. Robshaw, *Differential Cryptanalysis, Key-Dependent S-Boxes, and Twofish*, submitted to NIST as AES comment, 2000.

D. Whiting, *Twofish Technical Report No. 4*, submitted to NIST as AES comment, 1999.

N.B. Most of these references are available from the NIST AES website.

3.2   MARS

The MARS design uses many different components. According to the MARS designers, all used components are easy to analyse (C. Burwick *et al*, MARS - a candidate cipher for AES). We have a different view on this. The analysis of the MARS design team contains several errors with respect to the bounds for resistance against linear and differential cryptanalysis (see Knudsen and Raddum). There are also some relations between the different outputs of the E-function (see Robshaw). Furthermore, the S-boxes of MARS do not fulfill all the design requirements that were put forward by the team (see Burnett *et al*). Whilst the observations do not lead to an attack on MARS, they show that there is no clear view on the security of the MARS structure and its security yet. It is expected that further similar errors in the design team's analysis of MARS will be uncovered.

The added security of the unkeyed rounds is not clear. For instance, it has been shown that a reduced version of MARS, with 16 unkeyed rounds and 5 keyed (core) rounds, can be attacked (in the case of 256-bit keys) (see Kelsey and Schneier).

There are impossible differentials over 8 rounds of the MARS core, that may lead to an attack (see Biham and Furman). 11-round attacks on the core of MARS were also presented.

The key schedule is suboptimal in several ways. It is a quite complex algorithm. The need to check for keys with long runs of consecutive ones or zeroes, implies that a timing

attack resistant implementation of the key scheduling is very slow, because for all values of the key, the setup should last as long as in the slowest case. On the other hand, the test for such long runs is necessary, as subkeys with long runs of ones or zeroes may lead to efficient attacks on MARS.

The two least significant bits of the round keys that are used in the multiplication, are always set to 11. This is done in order to avoid weak keys, as exist in IDEA (see Daemen). However, setting these two key bits to 11 implies that there are always 2 inputs that go unchanged through the multiplication regardless of the subkey (all multiples of 2^{31}), and two others which have fixed output regardless of the subkey (odd multiples of 2^{30}).

We see that all components of MARS do not give the expected strength to the cipher, and there are questions whether the combination of all these components protect against all the weaknesses of the individual components.

Finally, we note that there are some interesting papers considering the pseudorandomness of the MARS (and other AES finalists). We do not believe that the pseudorandomness analysis performed so far is sufficient to contribute significantly to the selection of the AES.

In summary, the structure of MARS is too complex to evaluate in the short period of time of the AES process, but its complexity might lead to some weaknesses that may be uncovered at a later date.

MARS References

E. Biham and V. Furman, Impossible Differentials on 8 rounds of the MARS core, AES3, New York, 2000.

L. Burnett *et al*, *Effiecient Methods for Generating MARS-like S-Boxes*, Fast Software Encryption, New York, 2000.

C. Burwick *et al*, *MARS – A Candidate Cipher for AES*, 1998.

J. Daemen, Weak Keys for IDEA, Crypto 1993.

J. Kelsey and B. Schneier, *MARS Attacks!*, AES3, New York, 2000.

L. Knudsen and H. Raddum, *Linear Approximations to MARS S-Box*, submitted to NIST as AES comment, 2000.

M. Robshaw and Y. Lin, *Potential Flaws in the Conjectured Resistance of MARS to Linear Cryptanalysis*, submitted to NIST as AES comment, 2000.

N.B. Many of these references are available from the NIST AES website.

4   Performance Evaluation

Introduction

Whilst security is of course the main criterion for the future cryptographic standard, efficiency is also an important criterion. However, the wide range of target platforms, conditions of use etc., make efficiency very difficult to measure. In this section, we give an approach to this problem. We first propose a methodology for the performance evaluation of cryptographic primitives. We then review the research on performance carried out so far as part of the AES process and highlight the areas that have not received sufficient attention in this (largely voluntary) performance evaluation process. As we are considering generic cryptographic primitives, the comments in this section have more general applicability than those in the security evaluation section above.

4.1   Methodology

We now propose an evaluation process for the performance of a cryptographic algorithm. The first subsection considers the most important questions in building an evaluation framework. The second subsection then proposes some answers (or hints to answers) to these questions. Although we try to be as open and objective as possible, the suggested answers in this subsection inevitably partially reflect the authors' opinion.

4.1.1   Questions

1. *What are the performance requirements for a cryptographic primitive?*

   Depending on the platform (workstation, smart card, ASIC etc), different limited resources can have various importance.

   For software implementations, the following resources are considered:

   - time
   - code size
   - memory used

   For hardware implementations, the following resources are considered:

   - throughput
   - latency

- chip area

- power consumption

Moreover, the role of different aspects of the algorithm needs to be considered. The main aspects are:

- setup: this is the part of the algorithm done independently of the key and the data; it consists for example of constant tables (e.g. S-boxes) computation and filling;

- precomputations: this part can be done once for multiple encryptions with the same key; a good example is the key schedule;

- the primitive: this is the part that must be done for every use, that is, the block encryption itself.

Remark: it is worth noting that resource usage (computation time, memory , ...) may be asymmetric (encryption vs. decryption).

Clearly, the relative importance of these parts varies very much depending on the conditions of use. For example, the first two aspects are negligible if a large amount of data is to be encrypted under the same key on a machine with sufficient RAM, so different applications can have very different requirements. The relative importance of different applications is thus required to make an informed decision. Thus either information is needed or assumptions have to be made concerning the situations in which the algorithms are likely to be used.

2. *What if the performance depends on the input data?*

For some algorithms, specific input can lead to a very fast response, or a very slow, especially for the algorithm setup and precomputations (MARS key schedule or RSA modulus generation).

Are we interested in the average performance, or the worst case performance, or the worst not-too-improbable case?

3. *What precision can we hope for?*

Execution time, code size, etc. depend highly on the platform (CPU type, clock frequency, etc.). However, several other factors will also influence results. For example:

- hardware "details": cache size, etc.

- environment: operating system, etc.

- load of the machine at the time tests were performed

- compiler

- many time/memory tradeoffs (e.g. loop unrolling) are also possible.

Several of these issues are addressed in following sections. However, these factors imply that it is not possible to obtain exact figures regarding performances, and thus that small performance differences between candidates should be ignored. It is however often possible to give:

- (approximate) mean performances in given conditions

- lower bound on some resources, given limitations on others (e.g. lower bound on RAM usage, given some limitation on code size, including lookup tables)

4. *Which test platforms should be used?*

   Some scenarios do not occur on all platforms. The information or assumptions about the likely situations of use should indicate the relative importance of the different types of environments.

   (a) *Hardware Platforms*

   "Small" hardware differences (e.g. cache size) may have an important impact on performance. Comparisons must therefore be performed using exactly the same hardware. It is for example meaningless to make a precise comparison of implementations tested independentently on Pentium II and Pentium Pro, even if the results are scaled to eliminate differences in clock frequencies. It is (even more) meaningless to compare 8051 and 6805 implementations.

   Pure hardware implementations (e.g. ASIC) are very difficult to evaluate, because the design and optimisation of such devices is a very long and difficult operation.

   (b) *Software Platforms*

   Regarding the programming language, the question is twofold:

   i. Which programming languages should be considered (assembly language, ANSI C, Java, ...)?

   ii. Which specific compiler implementation should be used? Should the source code be allowed to test which compiler is used?

As we discussed above, the operating system may also have some importance. Other questions are :

- which extensions of ANSI C should be authorised?
- should assumptions on endianness be allowed?
- should the code be tuned for some platforms? For example the Mars C code of Gladman is 3 times slower than OptCCode on Intel486DX2 and 2 times faster on Pentium II.

5. *What are the implementation hypotheses?*

   Several other implementation decisions must be made for a coherent comparison.

   The first of these is related to "physical" attacks (timing attack, simple and differential power analysis, memory probing,...). This decision is not easy to make, as the state-of-the-art of this quite new branch of cryptanalysis is quickly evolving:

   - resistance against timing attack is not too difficult to specify: a strong, sufficient condition is to ask for the implementation to be constant time;
   - on the other hand, what has to be done to be DPA-resistant is not very clear yet. Several countermeasures have been proposed, that make the attack more difficult, but no "definitive" solution is currently known.

   However, for the comparison to be fair, one must precisely specify what attacks the implementation must resist, and with which strength.

   Another implementation question is whether self-modifying code is allowed or not.

6. *Who performs the implementations?*

   We face a dilemma in answering this question.

   - Relying on authors' implementations, risks favouring good implementors over good cryptographers.
   - Performing independent implementations, risks missing some useful "tricks", that require a deep understanding of the algorithm.

7. *How will the performances be measured?*

   In a multi-process environment, the running time of a cryptographic primitive depends on the load of the machine. We might want to measure the performance when

11

no context switch nor cache miss occurs, but a primitive can be such that, for every real-life usage, these situations cannot be avoided. There are also different measures for assessing the performance, such as the best performance or the mean performance.

8. *How will the results be combined?*

It is obvious that many different tests have to be performed. The performance evaluation will therefore not end up with a single figure summarising each algorithm's performance, but rather with an array of figures depending on the platform, time/memory trade- off, etc. The results may be conflicting, that is one algorithm may have good performance in one environment, and bad performance in another. However, for a decision to be made, these various results must somehow be combined. An important question is thus: how will the different test results be combined?

4.1.2 Suggested Answers

1. *What are the performance requirements for a cryptographic primitive?*

Limited resources are only a problem in very constrained environments (typically, smart cards); on more powerful platforms, code size, memory size etc. are far less important provided:

- they remain in reasonable limits (e.g. 32 GB table is clearly unrealistic, even on a powerful machine)
- their impact on performance is taken into account (e.g. a big table will not fit in the cache)

Regarding the relative importance of different subparts, a few "typical" scenarios, representative for practical applications, should be set up; we suggest

- frequent rekeying, where the key changes every block
- occasional rekeying, where the key changes every few blocks
- no rekeying, where the same key is used for large quantities of data

Besides these scenarios, flexibility (that is, ability to adapt to other situations) should be considered as an important factor.

2. *What if the performance depends on the input data?*

Two numbers should be computed:

12

- the performance in the mean case.
- with the exception of the most improbable worst cases (a proportion of 2^{-40} for example), the performance for the bad input should be computed.

The first number corresponds to a generic implementation, the second to an implementation protected against timing attacks.

3. *What precision can we hope for?*

Precision must not be overestimated. Even if full precision were achievable, a 10% speed difference is completely meaningless.

4. *Which test platforms should be used?*

(a) *Hardware Platforms*

For the tests to be representative, we suggest considering performances on at least:

- smart cards: it is difficult to decide exactly which ones, because differences can have a big impact (e.g. the addressing mode on the 8051 is much more powerful than on the 6805). We propose:
  - one low-cost smart card: the most common (6805, 8051) are probably the best choices
  - one powerful smart card, equipped with a cryptoprocessor.
  - one powerful smart card, not equipped with a cryptoprocessor (for example, some of the JAVA smart cards, which are likely to become popular, are not equipped with a cryptoprocessor);
- home PC (i.e. 32-bit): the official AES test platform (Pentium Pro), or a Pentium II, is probably a good choice
- a more powerful machine, based on a 64-bit architecture (representative of the "machine of the future"). It may be more difficult to choose a reference machine here, because the ones that are currently on the market (Alpha, UltraSparc, PA-RISC and MIPS) are quite different, and it is difficult to predict which one will "win the market" in future years. In view of this, flexibility MUST be considered as a major advantage.
- hardware (ASIC), or semi-hardware (FPGA): this test is however much more difficult to conduct, because of the difficulty of design.

As discussed above, comparisons should be made using exactly the same hardware for each algorithm.

(b) *Software Platforms*

For the comparison to be meaningful, the same OS should be used among candidates.

On the other hand, it is our belief that any specific compiler (i.e. the fastest for each candidate on the particular hardware) should be used. The reason of this is simple: if AES is faster with the XYZ compiler, everybody will use a XYZ-compiled AES. This is not always true for hardware (few people will buy a new computer just to speed-up AES) or OS, but changing compiler (or having the code compiled on a machine on which XYZ is installed) is an easy process.

We also believe that the importance given to high-level languages (e.g. C) implementations should not be overestimated: if speed is really important, assembly will always be used. For high-level languages, we should just consider the order of magnitude, reflecting the speed that can be achieved with "small programming investment".

For speed-critical applications, assembly language optimisations should be considered.

Some authors argued that Java implementations may help illustrate each candidate's "inherent" speed (that is, without exploiting "tricks" specific to a given architecture). This comparison seems however very difficult, as is illustrated by the big difference between the conclusions of Dray and Sterbenz (AES3 papers), due to the fact that Sterbenz tried to put the same effort in optimising each Java implementations. Moreover, Java performance is strongly dependent on the platform (JIT compiler and underlying hardware). We do not believe performance comparison of Java implementations to be more meaningful than implementations in C.

5. *What are the implementation hypotheses?*

Resistance against timing attacks should not slow down the implementation too much.

Resistance against power analysis and related attacks is still an open research problem. We do not have precise suggestions regarding strength against SPA and DPA. Maybe an interesting starting point can be [Daemen, Peters and Van Assche] and

14

[Messerges], presented at FSE2000: the first one proposes some techniques to protect a bitslice cipher against these attacks and sketches a formal definition of DPA-resistance; the second proposes countermeasures, not limited to the bitslice case. Moreover, such attacks depend on the platform: they are much more likely against smart cards than against servers, and future smart cards might be protected against them.

Due to its difficult and non-standard character, we believe the use of self-modifying code in making AES candidate comparisons is not appropriate.

6. *Who performs the implementations?*

   The best solution is probably to use a mixed approach, in which independent implementations are performed by a single person (or by a small group with strong interaction). This should ensure that difference in performance are not caused by a difference in programming skills.

   Note that the difficulty of realising a fast and accurate implementation should also be considered: a speed improvement that is so complicated that it can only be programmed by the algorithm's authors is of dubious value. Difficult tricks for the implementation should be fully explained.

7. *How will the performances be measured?*

   Except for a server environment with many context switches, the measurements should avoid taking the influence of multitasking (interruptions, context switches, cache misses) into account.

8. *How will the results be combined?*

   In our belief, the winner should be:

   - at least acceptable on each platform;
   - perform above the average on most platforms where there is a significant difference;
   - flexible, because test platforms only reflect current architectures, and it is difficult to predict what future machines will look like.

   We do not have a precise combination rule to propose, but it seems obvious that some weighting should be used when combining. This should reflect the relative importance of different platforms (e.g. what is the importance of speed on a smart card with respect to speed on a PC?).

4.2 A Survey of Performance Results for AES

There have been many C implementations of AES candidates, mainly optimised for Pentium processors. These have been tested on other CPUs. There are also some assembly implementations, FPGA and ASIC simulations of VDHL implementations and some smart card implementations.

We now list papers and web pages that give AES performance results. We summarise the scenario considered in the paper, rather than their results. (The results can be found in the paper.) Most papers are available online.

References

AES3   http://csrc.nist.gov/encryption/aes/round2/conf3/aes3agenda.html
AES2   http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm
AES R1   http://csrc.nist.gov/encryption/aes/round1/pubcmnts.htm
FSE2000 http://www.counterpane.com/fse-papers.html

4.2.1 C/Assembly on Workstations

1. Worley, Worley, Christian, Worley (AES3)
   "AES Finalists on PA-RISC and IA-64: Implementation & Performance"
   128 bit keys, hand optimised assembly
   eliminates cache and system effects (best running time)
   simulators for IA-64
   source code not available

2. Schneier, Whiting (AES3)
   "A Performance Comparison of the Five AES Finalists"
   128/192/256 bits key
   key setup, encryption
   compilation of other's results, some implementation done, fastest chosen various architectures (Pentium, PPro, PA8200, IA64, etc.)
   performance/security tradeoff (number of rounds)
   performance for maximal insecure variant
   smart cards RAM requirement

3. Bassham, NIST (AES3)
   "Efficiency Testing of ANSI C Implementations of Round 2 Candidate Algorithms"

16

for the AES

128/192/256 bits key

key setup, encryption, decryption (ECB)

OptCCode from submitters

various architectures (Pentium Pro/II/III, UltraSparc, R12000)

timing/cycle count; median, standard deviations

4. NIST (AES2)

"Timing and cycle count measurements of implementations in ANSI C, for multiple platforms and compilers"

Statistics are done for key setup, encryption, and decryption, based on the median value for a serie of 1000 measurements.

Treats also speed and memory measurements for Java implementations.

5. Schneier, Kelsey, Whiting, Wagner, Hall, Ferguson (AES2)

"Performance Comparison of the AES Submissions"

Concerns all the 15 initial candidates.

Platforms used: 8-bit Smartcard CPU, 32-bit CPU (Pentium/Pentium Pro), 64-bit CPU.

128 bits key

Measurements for: Key Setup, encryption, encryption for different plaintext sizes (1Ko,2ko,4Ko,...), algorithm used as an hash function.

Count the number of clock cycles. Results summarised by retaining average number.

Compare RAM requirements.

6. Aoki, Lipmaa (AES3)

"Fast Implementations of AES Candidates"

Comparison on a Pentium II, with special attention paid to MMX advantages

Refuse implementation "tricks" (self-modifying code, key-specific static variables, etc.).

Assembly-language implementations.

Does not consider decryption, nor key schedule.

Measures speed by running the test 500 times and taking the minimum result.

7. Lipmaa (AES2)

"AES Candidates: A Survey of Implementations"

Mainly a synthesis of performances for implementations already done (concerns all initial candidates, except Loki97 and Magenta).
Only independent implementation: Rijndael, on Pentium II and UltraSPARC II (Sun C).
Count the number of cycles per 128-bit block.

8. Gladman (AES2)
   `http:www.btinternet.com/~brian.gladman/cryptography_technology/aes/`
   "Implementation Experience with AES Candidate Algorithms"
   One implementor, C optimised for VC++ on Pentium Pro
   key setup, encryption and decryption
   source code available

9. Weiss, Binkert (AES3)
   "A comparison of AES candidates on the Alpha 21264"
   Gladman's implementation for Pentium Pro, tested on another machine.
   assembly implementation for Rijndael.
   encryption of multiple independent blocks simultaneously.
   measures done using cycle count register.

10. Osvik (AES3)
    "Speeding up Serpent"
    Implementation of Serpent on various x86-type platforms (486, Pentium, etc.).
    Special attention paid on optimisation of the s-boxes implementation.
    Implementation made in C except for critical routines, which are directly implemented in assembly language.

11. Biham (AES2)
    Comparison of speeds on well-known laptop.

4.2.2   Java

1. Dray, NIST (AES3, update of round 1 comments)
   "NIST Performance Analysis of the Final Round Java AES Candidates"
   JDK1.3beta compiler on Pentium Pro
   submitter's Java code
   key setup, encryption, decryption time
   implementation difficulty ratings

2. Sterbenz, Lipp (AES3)
   "Performance of the AES Candidate Algorithms in Java"
   Symantec Visual Cafe 2.5a JIT compiler on Pentium Pro
   one implementor (Sterbenz)
   key setup, encryption, decryption

3. Folmsbee (AES2)
   "AES Java Technology Comparisons"
   Despite his title, this paper essentially deals with the security criterion of avalanche.
   It's not very clear whether they perform independent implementations.
   Concerns all the 15 initial candidates
   Speed measurements in kbit/s. Platform used: Sun's Ultra Enterprise 2 (UltraSparc
   processor at 200 MHz, 256 Mb of RAM). Java Technology Interpreter, JIT compiler.
   Ram usage estimations by examining the source code and counting the number of
   variable declarations.

4. Aoki (AES round 1 comments)
   "Java performance of AES Candidates"
   Measurement of the speed of the Java Implementations of the 15 initial candidates
   on a (very!) wide range of computers and of JVM. Results reported on 50 pages of
   tables...
   Measurements performed: 1-block Encryption/Decryption, Key Setup, Large block
   Encryption/Decryption.

4.2.3   Smart Card or other Dedicated Processors

1. Hachez, Koeune, Quisquater (AES2)
   "cAESar results: Implementation of Four AES Candidates on Two Smartcards"
   Smart Cards (8 bit and 32 bit)
   Independent implementation (except for Rijndael on 8051)
   No physical attacks taken into account
   Assembly language
   Key schedule and encryption simultaneously

2. Chari, Jutla, Rao, Rohatgi (AES2)
   "A Cautionary Note Regarding Evaluation of AES Candidates on Smart Cards"

19

Presents a DPA against a naive implementation of Twofish on 6805, and overviews how similar attacks could be performed against the other candidates.

3. Keating (AES2, updated for round 1 comments)
   "Performance Analysis of AES candidates on the 6805 CPU core"
   Implementation of Crypton, RC6, Rijndael and Twofish on a Motorola 6805 series 8-bit architecture (independent implementation, except for Twofish).
   Evaluation in a simulation environment, so the number of rounds executed can easily be counted.
   Measures done for single 128-bit block encryption and key scheduling (128-bit key). ROM and RAM requirements evaluated ; the algorithms were implemented to fit within 64/128 bytes RAM (depending on whether key is scheduled into ROM or not) and 1024 bytes ROM (flexibility was allowed where this would cause a large speed penalty).

4. Sano, Koike, Kawamura, Shiba (AES3)
   "Performance Evaluation of AES Finalists on the High-End Smartcard"
   High-End Smart Card (8 bit with crypto coprocessor)
   Protected against timing attacks
   Assembly language
   Key schedule + encryption simultaneously

5. Gladman (AES3 Rump)
   ARM

6. Wollinger, Wang, Guajardo, Paar (AES3)
   "How Well Are High-End DSPs Suited for the AES Algorithms?"
   High-End DSP : TMS320C6x
   Independent implementation, based on either official code or Gladman's code.
   C, plus some assembly optimisations and "Intrinsic functions" (allow C code to access hardware capabilities, see paper for more details).
   Considers sequential and parallel implementation.
   Also considers memory usage, but implementation was not aimed at keeping it low.

7. Clapp (AES3 Rump)
   TriMedia VLIW Media-processor

4.2.4 Reconfigurable

1. Elbirt, Yip, Chetwynd, Paar (AES3)
   "An FPGA Implementation and Performance Evaluation of the AES Block Cipher
   Candidate Algorithm Finalists"
   Language based (VHDL). Simulation (?)
   High-end FPGA (XCV1999BG560-4). 128 bit data stream, 12288 CLB.
   Simulation/Xilinx : best throughput (for each pipelining and unrolling: highest frequency).
   Encryption.

2. Weaver, Wawrzynek (AES3)
   "A Comparison of the AES Candidates Amenability to FPGA Implementation"
   Language based (VHDL). Estimation.
   Modern mid-sized FPGA (XC2S50 to XCV200), 1500 to 5000 CLB.
   Estimation of maximal pipeline to run at 50 MHz. Cryptographic core.

3. Gaj, Chodowiec (AES3)
   "Comparison of the hardware performance of the AES candidates using reconfigurable hardware"
   FPGA Implementation
   Language based (VHDL). Simulation.
   High-end FPGA (XCV1999BG560-6) + mid-sized FPGA (XC4028/4036 and XC4085).
   Simulation/Xilinx report : best throughput.
   Encryption/Decryption. Resource sharing.

4. Dandalis, Prasanna, Rolim (AES3 Rump)
   FPGA Implementation

4.2.5 Hardware

1. Ichikawa, Kasuya, Matsui (AES3)
   "Hardware Evaluation of the AES Finalists"
   VHDL, on Mitsubishi Electric's 0.35 micron CMOS ASIC design library.
   128-bit key version.
   Simulation only; authors themselves acknowledge much better performances could
   be achieved if true hardware was done.
   Not intended to keep hardware size small (loops unrolled, ...).

No pipeline.
Results summarised by retaining worst case.

2. Weeks, Bean, Rozylowicz, Ficke (AES3)
   "Hardware Performance Simulations of Round 2 AES Algorithms"
   ASIC, but simulation only
   Considers 2 scenarios: iterative (medium speed, small transistor count) and pipelined (high speed, much resource used).
   Non optimal implementation, but fair comparison (same effort on each candidate, cross-validation.
   Gives estimates for area + transistor count, speed for each subpart of algo, considering each key size.
   Sources are made available.
   Uses some "unfair" hypotheses: for example, placement of some registers was uniformised among candidates rather than optimised.

3. Satoh, Ooba, Takano, D'Avignon (AES3)
   "High-Speed MARS Hardware"
   Simulation only, up to the chip level
   Using CMOS technology, but not up-to-date
   Performance measurements: Nominal case and Worst case

4.2.6 Implementation Attacks

1. Daemen, Rijmen (AES2)
   "Resistance Against Implementation Attacks: A Comparative Study of the AES Proposals"
   Quite vague: reviews, for the 15 candidates, the basic operations they are based on, and estimates, based on this information, how difficult it would be to make the candidate immune to power attacks.

2. Biham, Shamir (AES2)
   "Power Analysis of the Key Scheduling of the AES Candidates"
   Proposes a power analysis aimed at the key schedule part of the algorithm.
   Quite vague (sketch of) attack, based on the discovering of hamming weight of key-related parts. Presents general issues on how the key schedule can be isolated and then reviews the 15 candidates, briefly discussing how hamming weight can be found

and the consequences this would have for the algorithm.

3. Daemen, Peeters, Van Assche (FSE2000)
"Bitslice ciphers and Power Analysis Attacks"
Bitslice ciphers and Power Analysis techniques for protection, applicability to Serpent.
Proposes some techniques to protect a bitslice cipher (only Serpent is thus concerned) against DPA, and sketches a formal definition of DPA-resistance.
The technique is only briefly sketched in the Serpent case.

4. Messerges (FSE2000)
"Securing the AES Finalists Against Power Analysis Attacks"
Proposes implementations of AES candidates that are (claimed to be) DPA-safe, based on the masking of immediate values, and evaluates the speed, RAM and ROM usage of the secured implementations (for encryption only) for an ARM-based processor.

4.2.7 Surveys

1. Clapp (AES2)
"Instruction-level Parallelism in AES Candidates"
The aim of this paper is to evaluate how the speed of algorithms varies as a function of the resources available in the CPU.
For this purpose, the author considers a real-life CPU: the Philips Trimedia TM-1100, and parameterise it by the number of instruction-issue slots. Using a parameterised C-compiler, he can therefore explore the performances of candidate algorithms on hypothetical machines. A Critical-path analysis is also performed, allowing to compute a theoretical upper-limit on performance. The algorithms studied are Crypton, E2, Mars, RC6, Rijndael, Serpent, and Twofish.

2. Graunke (AES round 1 comments)
"Yet Another Performance Analysis of the AES Candidates"
Aim: To get an idea of how candidates might perform on future microprocessors.
Platform: Idealised, general purpose architecture.
Two metrics of performance used: Critical path of encrypting one block, and total

number of micro-operations required.

3. `http://www.di.ens.fr/~granboul/recherche/AES/timings.html`
   Cycle count, best reported implementation for each cpu.
   Only the internal ciphering routine.

4. `http://www.tcm.hut.fi/~helger/aes/`
   Survey of the speed of the best known implementation of 13 candidates (measured in number of cycles / block), on various cpus and compilers.
   Treats only the internal ciphering routine.

4.3 Areas requiring Further Investigation

- No protocol for testing the efficiency.

- Most of the code is unpublished.

- No way to combine conclusions.

A Dependence tests

Introduction

In this appendix, we examine the properties of the five 128-bit block ciphers MARS, RC6, Rijndael, Serpent, and Twofish with respect to the following four dependence criteria:

1. the average number of output bits changed when changing 1 input bit;

2. the degree of completeness;

3. the degree of avalanche effect;

4. the degree of strict avalanche criterion.

This examination is carried out for varying numbers of rounds, from 1 round to the full number of rounds. We consider a key size of 128 bits; the results for 196- and 256-bit-keys should be similar. For all 5 algorithms, the values of (1)–(4) are stable unless the number of rounds is very small.

Definitions

In this section, we will state the definitions of (2), (3) and (4), as given in J.-P. Boly, "Dependence Test", RIPE document Tools–P10–7, 1990.

For a vector $x = (x_1, \ldots, x_n) \in (\mathrm{GF}(2))^n$, the vector $x^{(i)} \in (\mathrm{GF}(2))^n$ denotes the vector obtained by complementing the i-th bit of x (for $i = 1, \ldots, n$). The *Hamming weight* $w(x)$ of x is defined as the number of nonzero components of x. A function $f : (\mathrm{GF}(2))^n \to (\mathrm{GF}(2))^m$ of n input bits into m output bits is said to be *complete*, if each output bit depends on each input bit, i.e.

$$\forall\, i = 1, \ldots, n \;\; \forall\, j = 1, \ldots, m \;\; \exists\, x \in (\mathrm{GF}(2))^n \text{ with } (f(x^{(i)}))_j \neq (f(x))_j.$$

A function $f : (\mathrm{GF}(2))^n \to (\mathrm{GF}(2))^m$ has the *avalanche effect*, if an average of $\frac{1}{2}$ of the output bits change whenever a single input bit is complemented, i.e.

$$\frac{1}{2^n} \sum_{x \in (GF(2))^n} w(f(x^{(i)}) - f(x)) \;=\; \frac{m}{2} \qquad \text{for all } i = 1, \ldots, n.$$

A function $f : (\mathrm{GF}(2))^n \to (\mathrm{GF}(2))^m$ satisfies the *strict avalanche criterion*, if each output bit changes with a probability of $\frac{1}{2}$ whenever a single input bit is complemented, i.e.

$$\forall\, i = 1, \ldots, n \;\; \forall\, j = 1, \ldots, m \;\; \Pr((f(x^{(i)}))_j \neq (f(x))_j) = \frac{1}{2}.$$

The *dependence matrix* of a function $f : (\mathrm{GF}(2))^n \to (\mathrm{GF}(2))^m$ is an $n \times m$-matrix A whose (i,j)-th element a_{ij} denotes the number of inputs for which complementing the i-th input bit results in a change of the j-th output bit, i.e.

$$a_{ij} \;=\; \#\{x \in (\mathrm{GF}(2))^n \,|\, (f(x^{(i)}))_j \neq (f(x))_j\}$$

for $i = 1, \ldots, n$ and $j = 1, \ldots, m$.

The *distance matrix* of a function $f : (\mathrm{GF}(2))^n \to (\mathrm{GF}(2))^m$ is an $n \times (m+1)$-matrix B whose (i,j)-th element b_{ij} denotes the number of inputs for which complementing the i-th input bit results in a change of j output bits, i.e.

$$b_{ij} \;=\; \#\{x \in (\mathrm{GF}(2))^n \,|\, w(f(x^{(i)}) - f(x)) = j\}$$

for $i = 1, \ldots, n$ and $j = 0, \ldots, m$.

Of course, unless the number of input bits is small, it is impossible to compute the dependence and distance matrices for all possible inputs. Therefore, one usually considers

a "suitable" number of randomly chosen inputs. The dependence and distance matrices are then defined as follows:

$$a_{ij} \;=\; \#\{x \in X \mid (f(x^{(i)}))_j \neq (f(x))_j\}$$

for $i = 1, \ldots, n$ and $j = 1, \ldots, m$ and

$$b_{ij} \;=\; \#\{x \in X \mid w(f(x^{(i)}) - f(x)) = j\}$$

for $i = 1, \ldots, n$ and $j = 0, \ldots, m$, where X is a "suitable" randomly chosen subset of $(\mathrm{GF}(2))^n$.

Let us now assume that we have computed the dependence matrix A and the distance matrix B of a function $f : (\mathrm{GF}(2))^n \to (\mathrm{GF}(2))^m$ for a set X of inputs, where X is either $(\mathrm{GF}(2))^n$ or a random subset of $(\mathrm{GF}(2))^n$. The *degree of completeness* of f is defined as

$$d_c \;=\; 1 - \frac{\#\{(i,j) \mid a_{ij} = 0\}}{nm}.$$

The *degree of avalanche effect* of f is

$$d_a \;=\; 1 - \frac{\sum_{i=1}^{n} \left| \frac{1}{\#X} \sum_{j=1}^{m} 2jb_{ij} - m \right|}{nm}.$$

The *degree of strict avalanche criterion* of f is defined as

$$d_{sa} \;=\; 1 - \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} \left| \frac{2a_{ij}}{\#X} - 1 \right|}{nm}.$$

For the function f to have good degrees of completeness, avalanche effect, and strict avalanche criterion, the numbers d_c, d_a, and d_{sa} must satisfy

$$d_c = 1, \quad d_a \approx 1, \quad d_{sa} \approx 1.$$

AES Finalists

We now assess the five AES finalists MARS against the dependence criteria defined above. For each finalist, we consider 10000 randomly chosen inputs encrypted under a single randomly chosen 128-bit key.

MARS

MARS encryption consists of

- key addition plus 8 rounds of unkeyed forward mixing, say fm0, ..., fm7;

- 8 rounds of keyed forward transformation, say t0, ..., t7, plus 8 rounds of keyed backwards transformation, say t8, ..., t15;

- 8 rounds of unkeyed backwards mixing, say bm7, ..., bm0, plus key subtraction.

We obtained the following values for (1)–(4):

Rounds			(1)	(2)=d_c	(3)=d_a	(3)=d_{sa}
fm0–fm7	t0–t15	bm7–bm0	64.003245	1.000000	0.999294	0.992034
fm0–fm7	t0–t11	bm7–bm0	63.999245	1.000000	0.999177	0.992050
fm0–fm7	t0–t7	bm7–bm0	63.997880	1.000000	0.999299	0.992045
fm0–fm7	t0–t3	bm7–bm0	63.996656	1.000000	0.999293	0.992078
fm0–fm7	./.	bm7–bm0	64.000670	1.000000	0.999240	0.992091
fm0–fm6	./.	bm6–bm0	64.004832	1.000000	0.999313	0.992081
fm0–fm5	./.	bm5–bm0	63.999882	1.000000	0.999325	0.991974
fm0–fm4	./.	bm4–bm0	63.998637	1.000000	0.999292	0.991990
fm0–fm3	./.	bm3–bm0	64.003864	1.000000	0.999347	0.991998
fm0–fm2	./.	bm2–bm0	53.338305	0.921082	0.832921	0.824234
fm0–fm1	./.	bm1–bm0	29.646600	0.683838	0.463090	0.448975
fm0	./.	bm0	9.990280	0.268372	0.156098	0.132855
./.	./.	bm0	11.048362	0.322876	0.172631	0.149615
fm0	./.	./.	8.099524	0.260193	0.126555	0.104392

For 3 rounds of forward mixing, no rounds of keyed transformation, and 3 rounds of backwards mixing, the values of (1)–(4) become worse; for 2 rounds of forward mixing, no rounds of keyed transformation, and 2 rounds of backwards mixing, the values of (1)–(4) become significantly worse.

RC6

RC6 is a block cipher with 20 rounds.

No. Rounds	(1)	(2)=d_c	(3)=d_a	(4)=d_{sa}
20	63.994746	1.000000	0.999304	0.992092
16	63.991403	1.000000	0.999280	0.992041
12	64.002503	1.000000	0.999375	0.992082
8	63.996466	1.000000	0.999249	0.992089
7	63.996148	1.000000	0.999247	0.992006
6	64.008165	1.000000	0.999282	0.992014
5	64.000126	1.000000	0.999342	0.992085
4	63.918859	1.000000	0.998433	0.991595
3	60.308607	1.000000	0.942272	0.937900
2	41.847271	0.875000	0.653864	0.652114
1	14.291377	0.419739	0.223303	0.213237

For 4 rounds, the values of (1)–(4) become slightly worse; for 2 rounds, they decrease significantly.

Rijndael

Rijndael is an algorithm with 9 rounds plus a final round. We always performed the final round ("+1").

No. Rounds	(1)	(2)=d_c	(3)=d_a	(4)=d_{sa}
9+1	63.994571	1.000000	0.999246	0.991925
8+1	63.998811	1.000000	0.999198	0.991974
7+1	64.003631	1.000000	0.999311	0.992072
6+1	64.003147	1.000000	0.999296	0.992019
5+1	64.001554	1.000000	0.999344	0.992030
4+1	64.005684	1.000000	0.999296	0.992065
3+1	64.001791	1.000000	0.999350	0.992043
2+1	64.247014	1.000000	0.996140	0.991466
1+1	16.064059	0.250000	0.251001	0.247672
0+1	4.040025	0.062500	0.063125	0.059129

For 2 rounds plus the final round, the values of (1)-(4) become slightly worse; for 1 round plus the final round, they deteriorate significantly.

Serpent

Serpent is a cipher with 32 rounds.

No. Rounds	(1)	(2)=d_c	(3)=d_a	(4)=d_{sa}
32	63.995637	1.000000	0.999255	0.992080
24	63.991927	1.000000	0.999233	0.992003
16	63.994650	1.000000	0.999282	0.992051
8	64.000817	1.000000	0.999319	0.991991
7	64.006302	1.000000	0.999272	0.991986
6	63.999410	1.000000	0.999389	0.992065
5	64.002679	1.000000	0.999311	0.991946
4	63.987744	1.000000	0.999254	0.992032
3	63.998755	1.000000	0.999271	0.992002
2	57.368955	0.941406	0.896390	0.805473
1	12.131035	0.148926	0.189547	0.107760

For 2 rounds, the values of (1)–(4) decrease; for just 1 round, they decrease significantly.

Twofish

Twofish is an algorithm with 16 rounds.

No. Rounds	(1)	(2)=d_c	(3)=d_a	(4)=d_{sa}
16	63.996125	1.000000	0.999322	0.991944
12	63.997724	1.000000	0.999306	0.991966
8	63.995787	1.000000	0.999245	0.992022
4	63.994789	1.000000	0.999295	0.992041
3	63.948767	0.998047	0.996912	0.982727
2	48.357374	0.750000	0.754228	0.725825
1	16.929400	0.255859	0.264522	0.238981

For 3 rounds, the values of (1)–(4) become slightly worse; for 2 rounds, they deteriorate significantly.