From: maro@isl.ntt.co.jp
To: AESFirstRound@nist.gov
Subject: Comments on NIST's Efficiency Testing for Round1
AES Candidates
Date: Thu, 15 Apr 1999 23:09:50 JST
Sender: maro@sucaba.isl.ntt.co.jp

Dear AES Candidate Comments Secretariat,

I submitted the report titled
Comments on NIST's Efficiency Testing for Round1 AES Candidates
using PostScript. The file is generated with gzip and
uuencode. If you cannot print the file, I will help you.

Best regards,

/ NTT Laboratories
/ AOKI, Kazumaro
/ E-mail: maro@isl.ntt.co.jp

Comments on NIST's Efficiency Testing for Round1 AES Candidates

Kazumaro AOKI*

April 12, 1999

NIST presented "NIST's Efficiency Testing for Round1 AES Candidates" at the Second AES Conference, AES2. The presentation included a performance comparison using the submitted codes of all AES candidates. We think that the presentation failed to offer a fair comparison. We described the problems on the copy of slides distributed at AES2.

1. In Slide #29, NIST summarized Java Values. We found that the speeds of E2, HPC, RC6, and SERPENT are very different from the data as shown in Table 1. Please confirm the test once again.

Table 1: Rank Comparison of Java Encryption Efficiency Test

	NIST	DFC team[a]	Folmsbee[b]		Ours[c]
Algorithm			MCT	KAT	
E2	12	6	3	4	3
HPC	13	4	5	7	6
RC6	10	1	1	1	1
SERPENT	14	5	6	5	5

[a] Table 2 in "Report on the AES Candidates" in proceedings of AES2 on page 56. The results were obtained on UltraSparc-I with JDK-1.2 (JIT).

[b] Table 3 in "AES Java Technology Comparisons" in proceedings of AES2 on page 39. The results were obtained on UltraSparc with JIT.

[c] "Java Performances of AES Candidates (draft)" distributed at AES2. The results were obtained on Pentium Pro with JDK-1.1.7 (JIT).

2. In the "Reference" configuration (Slides #9–12), NIST measured one block encryption time, which includes NIST API overhead. The NIST API overhead of the code of each AES candidate differs according to the programmers' policy. A fair comparison should reflect the algorithms' efficiency only, i.e., should be done *without* NIST API.

*NTT Laboratories (maro@isl.ntt.co.jp)

A method of removing the NIST API overhead is as follows. Roughly speaking, the clock cycles required by `blockEncrypt()` can be written as follows

$$b \times n + a,$$

where b is the average time for one block encryption, n is the number of blocks for an encryption, and a is the required time for NIST API.

NIST required that `blockEncrypt()` encrypt just one block, but all submitted codes can encrypt several blocks. Slides #9–12 show $b + a$ but we need b. Thus, if NIST measures 2 block encryption time, $2b + a$, NIST will get b by computing $(2b + a) - (b + a) = b$, for example.

NIST also tested the encryption of 1MB data (Slide #21). Because we know the E2 cycles, we guess that the tests used only one `blockEncrypt()` call with `inputLen` as 65538×128 bits. Slide #21 shows "Kb/s" not "cycles," but we can easily calculate cycles ($= 65538b + a$) with the knowledge of the platform frequency (200MHz). We can then calculate the averaged cycles for one block encryption ($= b + a/65538$). Because a is smaller than $b + a$, and $b + a$ is smaller than 9401, which one encryption of HPC requires, $a/65538 \leq 9401/65538 \approx 0$. So, Slide #21 can be used to indicate block encryption time *without* NIST API overhead. NIST wrote that the NIST API has minimal impact (Slide #19). However, comparing Slides #16 with #21 shows that there are big differences as shown in Table 2. Note that some candidates are slower at 1MB encryption than one block encryption. We guess that cache effects are bigger than NIST API overhead for these candidates.

3. The Slide #15 compares the Key Setup (128-bits). The slide refers to Gladman's result. However, he did not optimize for key setup (by personal email). This comparison should be done much more carefully. That is, a simple ranking should be avoided for the following reason.

   `makeKey()` includes ASCII to binary conversion. Some candidates that have fast key setup require longer time for ASCII to binary conversion than key scheduling. The key scheduling of MAGENTA is clearly fastest, but the key setup for CRYPTON is faster than that for MAGENTA according to NIST's result.

4. In Slide #29, NIST summarized Java Values. According to the slide, E2 requires about 259KB for heap. It is the largest requirement among the candidates, however, because recent computers have a lot of memory, requiring 300KB memory is no problem.

   The submitted code of E2 was optimized for speed rather than memory. The code combines 2 s-box tables which requires 256KB memory for the s-box table. We also developed a 1 s-box table version which requires 1KB memory for the s-box table. Thus, it is possible to reduce memory by about 255KB at the cost of a 25% speed decrease. (See the AES electronic forum for E2, a related topic was discussed.)

Table 2: Number of clock cycles for 1 block encryption on Pentium Pro

	Slide#16	Slide#21	difference ($S16 - S21$)	ratio $((S16 - S21)/S16)$
CAST-256	2169	1832	+337	+16%
CRYPTON	579	669	−90	−16%
DEAL	3197	3148	+49	+2%
DFC	3491	3981	−490	−14%
E2	1523	996	+527	+35%
FROG	1611	2431	−820	−51%
HPC	9401	15629	−6228	−66%
LOKI97	3077	3782	−705	−23%
MAGENTA	9253	15440	−6187	−67%
MARS	807	639	+168	+21%
RC6	636	683	−47	−7%
RIJNDAEL	809	1116	−307	−38%
SAFER+	2095	2829	−739	−35%
SERPENT	1629	1825	−196	−12%
TWOFISH	973	1197	−224	−23%

5. Slides #7-8 summarize the compile options used in testing. The speed comparison should be done with the fastest speeds that the codes achieve. NIST fixed the compiler options, however, some candidates run faster than the executables generated with the 'best' compiler optimization option. Refer to `Makefile` of II-DES. The `Makefile` tests all combinations of optimization options.
(`ftp://ftp.iij.ad.jp/pub/IIJ/dist/IIDES/iides-1.1.tar.gz`)