Public Comments on the Draft Federal Information Processing Standard (FIPS) for the Advanced Encryption Standard (AES)

[in response to a notice in the February 28, 2001 Federal Register
(Volume 66, Number 40; pages 12762-12763)]

Commenter (#) (MM/DD/YYYY) **Page**

1. John Savard *(02/28/01)*

Date: Wed, 28 Feb 2001 17:12:42 -0700
To: AEScomments@nist.gov
From: John Savard <jsavard@powersurfr.com>
Subject: Key Sizes

By analogy with other block ciphers, I believe that Rijndael may be somewhat more secure for certain combinations of block sizes and key lengths than others.

For the 128-bit block size, the security "sweet spot" appears to be a 224-bit key.

Thus, I think it would be desirable to include the 160-bit and 224-bit key sizes in the standard at the least.

Note also that the availability of a 160-bit block size would assist interoperability with some hash function standards.

The reasons why I believe that a 224 bit key may possibly be more secure with Rijndael are these:

1) With a 224-bit key, the number of Mix Column steps in the cipher is a multiple of four. For Rijndael, this is almost like a Feistel cipher having an even number of rounds, rather than an odd number of rounds.

2) With a 224-bit key, there are seven 32-bit words in the key, while the block size is four 32-bit words. This makes the length of the shift register producing the subkeys as different as possible from the size of the subkey for each round, thus making the key schedule slightly less regular.

While there is no evidence of a weakness in Rijndael that a 224-bit key avoids, including it in the standard would enable that key length to be generally used in the event such a weakness were discovered, and the general arguments from the basic structure of Rijndael make that at least a plausible possibility.

John Savard

2. Don Johnson, Certicom Corporation *(03/01/01)*

From: "Don Johnson" <djohnson@certicom.com>
To: AEScomments@nist.gov
Date: Thu, 1 Mar 2001 11:42:53 -0500
Subject: Don Johnson on AES FIPS

NIST,
Thank you for your efforts in developing the AES FIPS and for selecting RIJNDAEL. Thank you also for specifying that the AES algorithms that meet the FIPS have a specific blocksize, keysize and rounds, as this will help ensure interoperability. Thank you again for continuing this effort with the modes of operation workshops which are intended to produce a modes of operation document that include various good ways of using AES.

I do see a use for yet another document on AES. Given that there are three AES keysizes and two Triple DES keysizes, it would seem very helpful for a (short) guidance document to aid users on various considerations for selecting among the keysizes and algorithms. I do not think this would need to be a standard, but it would be helpful if it were (at least) an official NIST published paper.

For example, at first blush, one might think that one would never need more than AES 128 bit keys, so some may say that is all that will ever be needed. However, the realization of significant size quantum computers (for example) could change that due to the square root attack which effectively halves the keysize. Others say that since the performance cost difference is (apparently) small in many cases, one should simply always use AES 256 bit keys and be done with it. As another example, the 64-bit blocksize of Triple DES can be a concern, but how much should this affect any decision to move to/use AES? I know there is a hope and expectation that new uses will use AES and old uses will transition to AES.

Such a guidance paper would also be a valuable input to the ANSI X9 effort to discuss keysize considerations and/or perhaps could tie into it. Food for thought. I am willing to assist with such a paper in any manner requested, but do believe that the most acceptance and benefit accrues if it is an official NIST guidance paper.

Don B. Johnson
Director of Cryptographic Standards
Certicom

3. Jean-Luc Cooke #1 *(03/01/01)*

Date: Thu, 1 Mar 2001 16:20:47 -0500
From: Jean-Luc Cooke <jlcooke@jlcooke.ca>
To: AEScomments@nist.gov
Subject: 4.2.1 typo
X-Mailer: Mutt 1.0i

3rd sentence.

If b_7=0, ... Should be b_8=0.

JLC
--
101 ways to get in touch with me:
<A HREF=http://jlcooke.net/contact>http://jlcooke.net/contact</A>

4. Jean-Luc Cooke #2 *(03/01/01)*

Date: Thu, 1 Mar 2001 16:23:02 -0500
From: Jean-Luc Cooke <jlcooke@jlcooke.ca>
To: AEScomments@nist.gov
Subject: 4.2.1 typo (corr.)
X-Mailer: Mutt 1.0i

B_7=0 is correct. My applogies.

JLC
--
101 ways to get in touch with me:
<A HREF=http://jlcooke.net/contact>http://jlcooke.net/contact</A>

5. John Myre, Sandia National Labs *(03/01/01)*

Date: Thu, 01 Mar 2001 16:27:38 -0700
From: "John Myre" <jmyre@sandia.gov>
Organization: Sandia National Labs
X-Mailer: Mozilla 4.61 [en]C-CCK-MCD SNL4.x (WinNT; U)
X-Accept-Language: en,pdf
To: AEScomments@nist.gov
Subject: Unclear notation in algorithm pseudo-code
X-WSS-ID: 16800560184432-01-01

Sirs,

In the draft AES FIPS, in figures 6, 13, and 16 (which have the pseudo-code for the encryption, decryption, and alternate decryption algorithms), the calls to AddRoundKey utilize a "C" idiom, pointer arithmetic, in selecting the subset of keys to add. While clear to C programmers, this is not likely to be clear to anyone else.

I would suggest some alternate notation. Perhaps the key schedule could be a 2-dimensional array, as in

```
word w [ Nr+1, Nb ]
...
AddRoundKey( state, w [round,*] )
```

or some such thing. Alternately, if the key schedule is kept as a single dimension array, one might use

```
AddRoundKey( state, w [(round * Nb) .. (round * (Nb+1) - 1)] )
```

Thank you for your AES efforts -

John Myre
Sandia National Labs
jmyre@sandia.gov

6. David Scott *(03/01/01)*

Date: Thu, 01 Mar 2001 17:00:30 -0700
From: David Scott <dscott@elpasonet.net>
Organization: retired
X-Mailer: Mozilla 4.72 [en] (Win98; I)
X-Accept-Language: en
To: AEScomments@nist.gov
Subject: Padding modes for AES

Dear Sir
below is text on RFC1423
"
http://www-users.aston.ac.uk/Connected/RFC/1423/rfc1423.txt
Balenson [Page 2]
RFC 1423 PEM: Algorithms, Modes and Identifiers February 1993

DES is defined in FIPS PUB 81 [3], and is equivalent to those provided in ANSI X3.106 [4] and in ISO IS 8372 [5]. The character string "DES-CBC" within an encapsulated PEM header field indicates the use of this algorithm/mode combination.

The input to the DES CBC encryption process shall be padded to a multiple of 8 octets, in the following manner. Let n be the length in octets of the input. Pad the input by appending 8-(n mod 8) octets to the end of the message, each having the value 8-(n mod 8), the number of octets being added. In hexadecimal, the possible paddings are: 01, 0202, 030303, 04040404, 0505050505, 060606060606, 07070707070707, and 0808080808080808. All input is padded with 1 to 8 octets to produce a multiple of 8 octets in length. The padding can be removed unambiguously after decryption.
"
The above is similar to the kind of padding we may be stuck with for the new AES. It has some major weaknesses that are easy to fix. The main weakness is that allows an attacker to throw away many
keys when trying to break a file since most keys will not lead to a file of the above form when using the wrong key for decyrption.

I would like to propose a better solution that does not weaken the encryption like the current standard method. I would be willing to present this type of padding if needed. Since I am a retired GS-13 engineer. My government service was at China Lake.

The concept is very simple. And involves a few very simple steps.

1) the starting file is an arbitrary 8-bit byte type of file. Convert it to an infinite file that is "finitely odd". An easy method of doing this is to look at each 8 bit byte in the file. If the trailing bytes are not (all values in HEX) "00" or "00" followed by one or more "80" then leave file alone and pretend it is followed by an infinite number of "zeroes". If the file

does have a trailing "00" or "00" followed by one or more "80" then add a trailing byte of "80" and then follow that by an infinte number of zeros.

Examples:
00 00 becomes 0000800000000000...
80 becomes 800000000...
00 80 becomes 0080800000000000...
30 80 becomes 3080000000000000...
40 00 becomes 4000800000000000...
23 12 becomes 23120000000000000...

note all files at this point have a last one "bit" followed by a tail of an infinite number of zeros.

2) convert this to a normal file the block size of the encryption. Lets assume the 16 byte block of the AES method. To convert group the infinte finitely odd file into blocks of 128 bits. Then ignore the infinite number of zero blocks that trail at end. At this point the last block can not be all zeros. So the next step is to look at the last block remaining. If it is of form "80 00 00 rest of block zero " at this poiint if last block is this check the next to last block until either you reach the start of file or a block that is not of form "80 00 00 rest of block zero" If the block your at is a totally "zero block" you drop the last block. If not you dont.

Examples here "zz" means rest of 16 byte block zero:
80 00 zz 80 00 zz 0000000000000000000...
becomes 80 00 zz 80 00 zz
00 00 zz 80 00 zz 0000000000000000000...
becomes 00 00 zz
23 00 zz 78 00 zz 0000000000000000000...
becomes 23 00 zz 78 00 zzzz

3) since all files have been "1 to 1" mapped to the desired block size do the encryption.

4) Now the files are encrypted and in the correct block size. At this point convert them back to finitely odd files. This is like step 1 but instead of 8 bit bytes you use 128 bit chunks. The token equaivalent to "00" is the block of 128 bits all zero. And the one equivalent to "80" is the block of 128 bits all zero except the first bit (left most) is a one.

5) convert this file doing the exact opposite of step one. This gives the final encrypted output file.

ONE SPECIAL PROPERTY of this kind of padding. Is that any 8-bit binary file can be thought of as an output file to the AES encryption. And encryption key test for any binary file. Will lead to a decrypted file that when encrypted with the method above goes back to the same file. I am willing to give a presentation on this if more explanation is needed.

Thank You
David A. Scott

7. Harald von Fellenberg, Sun Microsystems *(03/02/01)*

Date: Fri, 02 Mar 2001 13:49:49 +0100
From: Harald von Fellenberg <harald.von-fellenberg@Sun.COM>
Organization: Sun Microsystems
X-Mailer: Mozilla 4.75 [en] (X11; U; SunOS 5.7 sun4u)
X-Accept-Language: en
To: AEScomments@nist.gov, Harald.Von-Fellenberg@Sun.COM
Subject: typo p 17

Hello AES team:

I am only a physicist, but in my humble opinion the title of paragraph
4.2.1 in page 17 of the .pds document should read

4.2.1 Multiplication by x

and not

4.2.1 Multiplication by y

Reason: there is no mention of y in the paragraph

Please consider
thanks and regards

Harald von Fellenberg
--
****
Dr. Harald von Fellenberg
Senior Consultant Technology Strategy Office
Tel: ++41 1 908 9230 Sun Microsystems (Schweiz) AG
Fax: ++41 1 908 9001 Javastr. 2
Mobile: ++41 79 349 0393 CH-8604 Volketswil
mailto:harald.von-fellenberg@sun.com

8. Jean-Luc Cooke #3 *(03/04/01)*

Date: Sun, 4 Mar 2001 21:05:59 -0500
From: Jean-Luc Cooke <jlcooke@jlcooke.ca>
To: AEScomments@nist.gov
Subject: Round 2 vs. FIPS
X-Mailer: Mutt 1.0i

Can you explain the inconsistancy between:
Authors:
[1] (http://www.esat.kuleuven.ac.be/~rijmen/rijndael/rijndaeldocV2.zip, p.14)
NIST:
[2] (http://csrc.nist.gov/publications/drafts/dfips-AES.pdf, p.26)
[2] is not was we agreed to.
JLC
--
101 ways to get in touch with me:
<A HREF=http://jlcooke.net/contact>http://jlcooke.net/contact</A>

9. Jaechul Sung, CIST, Korea University *(03/13/01)*

From: "AshTray" <sjames@cist.korea.ac.kr>
To: <AEScomments@nist.gov>
Subject: Some errata in the draft of AES document
Date: Tue, 13 Mar 2001 14:16:02 +0900
X-Mailer: Microsoft Outlook Express 5.50.4133.2400

Hello.
I found some errata.

(1) In the Fig.13 of page 23, for round=Nr-1 step -1 to 1 should be replaced by for round=Nr-1 step Nr-1 to 1.

(1) In the Fig.16 of page 27, for round=Nr-1 step -1 to 1 should be replaced by for round=Nr-1 step Nr-1 to 1.

That's I found all.

Best Regards,
Jaechul Sung,
CIST, Korea Univ.,
SEOUL,KOREA

10. Kevin McGowan *(04/06/01)*

From: "Kevin McGowan" <kevinmcg3@hotmail.com>
To: AEScomments@nist.gov
Subject: AES algorithm
Date: Fri, 06 Apr 2001 13:19:27 -0500
X-OriginalArrivalTime: 06 Apr 2001 18:19:27.0820 (UTC)
FILETIME=[1DBE80C0:01C0BEC6]

I have recently been following the progress and developement of the new AES standard and although Rijndael is a very impressive pick, I feel that one much stronger algorithm has been overlooked. TecSec Inc., a Vienna,VA based cryptography company, produces an extremely high performance cryptographic algorithm called P2. P2 was developed by Ed Scheidt, the former CIA cheif of cryptography, with the utmost regard for producing a cutting edge algorithm. I strongly feel that P2 would outperform Rijndael, and highly recommend that P2 be tested before announcing Rijndael as the standard. I believe P2 to be a much stronger product and I believe the government would greatly benefit by looking into P2. - Kevin

---

Get your FREE download of MSN Explorer at http://explorer.msn.com

11. Miles Smid, CygnaCom Solutions *(04/17/01)*

From: "Miles Smid" <masmid@erols.com>
To: <AEScomments@nist.gov>
Cc: "Santosh Chokhani" <santosh.chokhani@entrust.com>
Subject: AES Comments
Date: Tue, 17 Apr 2001 19:02:55 -0400
X-Mailer: Microsoft Outlook IMO, Build 9.0.2416 (9.0.2910.0)
Importance: Normal

Attached you will find my comments on your Draft AES.
Miles Smid

Comments on the Proposed Advanced Encryption Standard (AES)

Miles E. Smid
CygnaCom Solutions
April 17, 2001

Overall, the standard is very well written. My comments are editorial in nature.

1. (Section 3.4, p10, last paragraph and Figure 4) It does not seem as though the text beginning with "Note that the four bytes…" and Figure 4 impart any new information beyond what was already covered by Figure 3 and is again covered in Section 3.5. I suggest eliminating the text beginning with "Note … and also Figure 4. Section 3.5 would then need to be modified to reference Figure 3 instead of Figure 4 and the a_i's changed to in_i's.
2. (Section 4.2, p12, hex representation of m(x)) I suggest representing m(x) as {01}{1b} instead of 1{1b}.
3. (Section 4.2, p13, line 11) Change "Moreover, it holds that" to "Moreover for any a(x), b(x), and c(x) in the field, it holds that".
4. (Section 4.2.1, p13) Change title of section from "Multiplication by y" to "Multiplication by x".
5. (Section 5.1.1, p17, item 1.) Change "Sec. 4" to "Sec. 4.2".
6. (Section 5.1.3, p20, line 2 and line 5) Change "Sec. 4.2" to "Sec. 4.3" in both places.
7. (Section 5.2, p22, Figure 12) Should "word temp = w[i– 1]" be written as temp = w[i– 1]"?
8. (Section 5.3.3, p25, line 3 and line 6) Change "Sec. 4.2" to "Sec. 4.3" in both places.
9. (Appendix B, p30, line 1) Change "key schedule using" to "key expansion using" and eliminate the use of "key schedule".
10. (Appendix B, p30, column 4 of table) Change "After SubByte()" to "After SubWord()".

12. Juan Asenjo, Thales e-Security *(04/20/01)*

From: "Asenjo, Juan" <Juan.Asenjo@thales-esecurity.com>
To: "'AEScomments@nist.gov'" <AEScomments@nist.gov>
Cc: "Epstein, Sandy" <Sandy.Epstein@thales-esecurity.com>,
"Jackson, Paul"
<Paul.Jackson@thales-esecurity.com>
Subject: Certification of AES Implementation
Date: Fri, 20 Apr 2001 18:38:02 -0400
X-Mailer: Internet Mail Service (5.5.2650.21)

NIST Representatives,
As an information systems security professional, I am writing to respectfully comment on the "Draft AES Algorithm Implementation" document posted in your web site. My concerns deals with the certification of AES algorithm implementations. The recently published Draft FIPS for AES states under Item 8. "Implementation:"

"Cryptographic modules that implement the algorithm specified in this standard shall conform to the requirements of FIPS 140-2."

Given that FIPS 140-2 is also in Draft, it is not clear how this affect the implementation of AES within a platform already certified under FIPS 140-1. The statement I quoted above appears to indicates that the algorithm must be implemented and evaluated in accordance with FIPS 140-2 before it can be offered as a certified product. If FIPS 140-2 is not yet available, how can a vendor insure availability of a compliant implementation by the time AES is formally published?

After discussing this issue with Mr. Jim Foti in your organization, I understand that the statement in the AES draft, was included in anticipation that 140-2 would have been approved by the time that AES was published. However, that not being the case, it does not take into account a window that will allow vendors accredited under FIPS 140-1 to receive 140-2 validations.

Given the statements made in the draft and the comments mentioned above, I believe clarification is in order, so that the validation process of AES implementations is clear to vendors.

I look forward to further guidance and thank you for the opportunity to comment on this important standard.

Sincerely,
Juan C. Asenjo
Product Marketing Manager
Network Security
Thales e-Security
tel: 954 846-5040
Fax: 954 846-3935
e-mail: juan.asenjo@thales-esecurity.com
web: www.thales-esecurity.com

13. Hideo Shimizu, Toshiba Corporation *(04/24/01)*

Date: Tue, 24 Apr 2001 15:43:12 +0900
From: Hideo Shimizu <hideo.shimizu@toshiba.co.jp>
X-Mailer: Mozilla 4.75 [ja] (Win98; U)
X-Accept-Language: ja,en,pdf
To: AEScomments@nist.gov
Subject: comment for draft of fips

Dear fips editor
I feel description of decoding is rather short than encoding.

- In page 25, we need inverse matrix of (5.2).
- We need description of inverse key expansion.

I think both items are useful for hardware implementer.
thanks

Hideo Shimizu, Toshiba Corporation

14. François Rousseau *(04/25/01)*

From: "Francois Rousseau" <frousseau58@hotmail.com>
To: AEScomments@nist.gov
Subject: Comments on Draft FIPS for AES
Date: Wed, 25 Apr 2001 11:32:42 -0400
X-OriginalArrivalTime: 25 Apr 2001 15:32:42.0863 (UTC)
FILETIME=[F82C77F0:01C0CD9C]

This message is in response to your request for comments on this draft Federal Information Processing Standard (FIPS) for the Advanced Encryption Standard (AES).

Here are some technical and editorial comments on this draft FIPS for AES:

a. Announcement, this draft FIPS for AES only briefly mentions the AES modes of operation in Appendix A when listing the current AES Object Identifiers (OIDs). Although it is recognized that NIST is expected to issue a guideline or recommendation in the spring of 2001 on the AES modes of operation, the final version of the FIPS for AES should explicitly be referring to this guideline or recommendation on the AES modes of operation. This would be similar to FIPS 46-3 about the Data Encryption Standard (DES) and the Triple Data Encryption Algorithm (TDEA), which explicitly refers to FIPS 81 on the DES Modes of Operations in its announcement section.

b. Announcement, Section 8 on Patents, with NIST effort to ensure that AES would be available royalty free worldwide, this section should only indicate that implementations of the algorithm specified in this standard are not expected to be encumbered by U.S. and foreign patents. However, in the event that NIST had recently uncovered patents or patents applications covering the Rijndael algorithm, this section would then have to indicate that NIST has obtained worldwide, nonexclusive, royalty-free rights under these patents for vendors to make, use and sell apparatus which complies with this standard.

c. Sections 1, 2.1, 2.2, 3.4, and 5, although this draft FIPS for AES only briefly mentions the Equivalent Inverse Cipher under Section 5.3.5 and Appendix D, it should probably be mentioned throughout in every instances where the Inverse Cipher is mentioned.

d. Section 2.1 or 2.2 should probably contain the expression "Rcon" from Figure 12.

e. Section 2.2, it should indicate that the transformation InvMixColumns() is also used in the Key Expansion routine as per Figure 16 on the Equivalent Inverse Cipher.

f. Section 2.2, the title of this section should be "Algorithm Parameters, Symbols, Transformations and Functions" since "Transformations" are covered under this section and "Terms" are already covered under Section 2.1. Note that this title change will also impact the brief description of Section 2.2 under Section 1. In addition, this section should also have a sentence similarly to Section 2.1 introducing these algorithm parameters, symbols, transformations and functions. Finally, the transformations SubBytes() and InvSubBytes() are not presented in the appropriate alphabetical order.

g. Section 5, the first two paragraphs of this section discuss the possible representations of the block size (Nb) and the key length (Nk) for the AES algorithm, however a comparable paragraph discussing the possible representations of the number of rounds (Nr) for the AES algorithm is missing. This additional paragraph should also explain why the currently specified possible representations of the number of rounds (Nr) (i.e. 10, 12 and 14) were determined to be adequate.

h. Figure 16, the additional equivalent inverse cipher pseudo code for the key expansion routine should probably use the term "round" instead of "rnd".

i. Section 6.3, except for the second sentence of Section 1, there seems to be general impression from this draft FIPS for AES that only the block size (Nb) could be modified in future reaffirmations of this standard, but not the key length (Nk) and/or the number of rounds (Nr). This is due to the fact that all the explicit references to Section 6.3 (i.e. in Sections 2.2, 3.4 and 5) are only made for the block size (Nb) and not the other two parameters. Similar references to Section 6.3, but for the key length (Nk) and/or the number of rounds (Nr) should be added in Sections 2.2, 3.1 and 5.

j. Section 6.3, the second sentence should indicate that future reaffirmations of this standard could include additions to the allowed values for those parameters but not "changes". Although the current values for the key length (Nk), block size (Nb), and number of rounds (Nr) explicitly allowed by this standard could be deprecated in future reaffirmations of this standard, for interoperability reasons they should never be changed. This section should also explicitly indicate if new values were allowed in future reaffirmations of this standard that new Object Identifiers (OIDs) would then be registered to explicitly identify them.

k. Appendix A should also include the Object Identifiers (OIDs) for the AES Counter mode since NIST has previously announced that the first version of its guideline or recommendation on AES modes of operation would include the four "basic" modes of operation and a counter mode.

l. Appendix D, the first sentence wrongly refers to "Nr" for the three AES key lengths instead of "Nk". In addition, the reference [6] for example vectors seems incorrect.

m. Appendix E, the footnote for reference [1] also refers to reference [1]. To avoid this circular reference, the footnote should read "...etc. - is available from this site."

I thank you for the opportunity to review this draft FIPS for AES. If you would like to discuss any of these comments, please feel free to contact me.

Best regards,
Francois Rousseau

Get Your Private, Free E-mail from MSN Hotmail at http://www.hotmail.com.

15. Sheeri Kritzer #1 *(05/07/01)*

```
Hello there,

While working through the Draft FIPS for the AES, in Section 4.2,
Multiplication, you state that

"For example, {57} * {83} = {c1}, because

(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1)  = x^13 + x^11 + x^9 + x^8 + x^7
+
                                x^7 + x^5 + x^3 + x^2 + x +
                                x^6 + x^4 + x^2 + x + 1 "

which is correct.  however, you then go on to state that that is equal
to

"x^13 + x^11 + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1"

which is incorrect.  In fact,
x^13 + x^11 + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x +
 x^6 + x^4 + x^2 + x + 1

actually equals x^13 + x^11 + x^10 + x^7 + x^2 + 1

which, mod (x^8 + x^4 + x^3 + x + 1)

equals

x^6 + x^3 + x^2 + 1

Respectfully yours,

Sheeri Kritzer
```

16. Sheeri Kritzer #2 *(05/08/01)*

Hi there!

I previously erroneously stated that Section 4.2 of the AES standard was incorrect. I was mistakenly doing decimal addition, not the XOR addition specified in Section 4.1.

-Sheeri Kritzer

17. Paulo Barreto *(05/09/01)*

On efficient implementation of InvMixColumns

Dear NIST AES team,

I would like to describe a technique to efficiently implement the `InvMixColumns` transform in storage-constrained, byte-oriented environments like smart cards (and possibly also hardware). Though straightforward, it's likely that most implementors will miss the technique and, hence the opportunity to produce optimized code.

Section 5.1 of the specification of RIJNDAEL submitted to NIST by its authors describes an efficient implementation of function `MixColumns` for 8-bit platforms. It benefits from the coefficients of the matrix used in `MixColumns`, which are as simple as possible, namely:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

Contrary to `MixColumns`, though, `InvMixColumns` involves quantities that make multiplication over GF(2^8) rather cumbersome, apparently implying that storage-consuming tables are needed (at least two 256-byte tables, namely, exponential and logarithm tables). In other words, it's difficult to implement multiplication by the matrix:

$$\begin{bmatrix} e & b & d & 9 \\ 9 & e & b & d \\ d & 9 & e & b \\ b & d & 9 & e \end{bmatrix}$$

However, this matrix can be written as:

$$\begin{bmatrix} e & b & d & 9 \\ 9 & e & b & d \\ d & 9 & e & b \\ b & d & 9 & e \end{bmatrix} = \begin{bmatrix} 5 & 0 & 4 & 0 \\ 0 & 5 & 0 & 4 \\ 4 & 0 & 5 & 0 \\ 0 & 4 & 0 & 5 \end{bmatrix} \bullet \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

The right factor of the above product is the same matrix used in `MixColumns`, and the left factor can be efficiently implemented, as illustrated below for one column:

$$\begin{bmatrix} 5 & 0 & 4 & 0 \\ 0 & 5 & 0 & 4 \\ 4 & 0 & 5 & 0 \\ 0 & 4 & 0 & 5 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x \oplus 4(x \oplus z) \\ y \oplus 4(y \oplus w) \\ z \oplus 4(x \oplus z) \\ w \oplus 4(y \oplus w) \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \oplus \begin{bmatrix} 4(x \oplus z) \\ 4(y \oplus w) \\ 4(x \oplus z) \\ 4(y \oplus w) \end{bmatrix}$$

Making use of the `xtime()` function described in the RIJNDAEL specification and in the FIPS draft, this amounts to the following pseudo-code:

```
tmp1 = xtime(xtime(x ⊕ z))
tmp2 = xtime(xtime(y ⊕ w))
x = x ⊕ tmp1
y = y ⊕ tmp2
z = z ⊕ tmp1
w = w ⊕ tmp2
```

This technique gets rid of any additional tables; it means that `InvMixColumns` can be implemented with the same resources as `MixColumns`, plus six exclusive-OR operations and four `xtime()` calls. Hardware implementations could further slightly benefit from combining the double `xtime()` above into a dedicated function x2time().

Best regards,

Paulo S. L. M. Barreto.
Chief Cryptographer
Scopus Tecnologia S. A.

18. Miles Smid *(05/17/01)*

This is a small point but I noticed that in Section 5.1.4 of AES where AddRoundKey () is defined, the function is defined for each round but not for the initial whitening. I believe that this could be easily corrected by changing the last line of the first paragraph of the section to state the bounds on "round" as

0 <= round <= Nr instead of 1 <= round <= Nr.

There are Nr rounds but the variable "round" has to cover Nr + 1 values in order for AddRoundKey () to be defined for all its applications.

Miles Smid
CygnaCom Solutions

19. N-W Wang et. Al. *(05/22/01)*

Concerns about the efficiency of data access and ambiguity in the AES Proposal

Neng-Wen Wang
Department of Engineering Science
National Cheng Kung University
Taiwan,R.O.C.

Yueh-Min Huang
Department of Engineering Science
National Cheng Kung University
Taiwan,R.O.C.

Chi-Sung Laih
Department of Electrical Engineering
National Cheng Kung University
Taiwan,R.O.C.

Abstract

NIST(National Institute of standards and Technology) announced that Rijndael was selected as the proposed AES(Advanced Encryption Standard) on Dec 2, 2000. Several Versions of programs have been proposed for AES implementation. However, there exist some concerned problems in section 4.1 of the AES proposal. One problem is that some paragraphs in this section are inconsistent, which may confuse the programmers, and the other one is the inefficient data access for the original matrix. Hence, we suggest the AES standard should adopt the transposed state matrix and transposed key matrix to eliminate this inconsistency problem and improve the efficiency of data access for the matrix.

1. Concern about efficiency of data access and ambiguity in AES Proposal

The Rijndael cipher is suited to be implemented efficiently on a wide range of processors and in dedicated hardware. However, there are some concerned problems on the efficiency of the data access and some ambiguity existed in the AES proposal. We will define the terminology in following paragraph, then describe our concerns in Rijndael Algorithm implementation.

1.1 The state and the cipher key

The different transformations operated on the intermediate cipher result are called the state. The state can be pictured as a rectangular array of bytes. This array has four rows, the number of columns is denoted by Nb and is equal to the block length divided by 32.

The cipher key is similarly pictured as a rectangular array with four rows. The number of columns of the cipher key is denoted by Nk which is equal to the key length divided by 32. These representations are illustrated in Fig 1.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Fig 1 Examples of state with Nb=6 and key with Nk=4

The input and output used by Rijndael at its external interface are considered to be one-dimensional arrays of 8-bit bytes numbered upwards from 0 to the 4*Nb-1. These blocks hence have lengths of 16,24, or 32 bytes and array indices in the ranges 0..15, 0..23, or 0..31. The cipher key is considered to be one-dimensional arrays of 8-bit bytes numbered upwards from 0 to the 4*Nk-1. These blocks hence have lengths of 16,24, or 32 bytes and array indices in the ranges 0..15, 0..23, or 0..31.

The cipher input bytes are mapped onto the state bytes in the order $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$,$a_{1,1}$,$a_{2,1}$,$a_{3,1}$, $a_{0,2}$,$a_{1,2}$,$a_{2,2}$,$a_{3,2}$, …, and The cipher key input bytes are mapped onto the state bytes in the order $k_{0,0}$,$k_{1,0}$,$k_{2,0}$,$k_{3,0}$, $k_{0,1}$,$k_{1,1}$,$k_{2,1}$,$k_{3,1}$, $k_{0,2}$,$k_{1,2}$,$k_{2,2}$,$a_{3,2}$, …. At the end of the cipher operation, the cipher output is extracted from the state by taking the state bytes in the same order.

1.2 Using Transpose Matrix to enhance the efficiency of data access

The Rijdael algorithm uses matrix to store the state and cipher key. All the encryption/decryption operations will be performed on these two matrices. For the implementation, if we use the original matrix orientation in Fig 1, the sequential word-access will become slow. To enhance the efficiency in data access, we can transpose the matrix as the one in Fig 2. We will give the explanation in the next paragraph.

$a_{0,0}$	$a_{1,0}$	$a_{2,0}$	$a_{3,0}$
$a_{0,1}$	$a_{1,1}$	$a_{2,1}$	$a_{3,1}$
$a_{0,2}$	$a_{1,2}$	$a_{2,2}$	$a_{3,2}$
$a_{0,3}$	$a_{1,3}$	$a_{2,3}$	$a_{3,3}$
$a_{0,4}$	$a_{1,4}$	$a_{2,4}$	$a_{3,4}$
$a_{0,5}$	$a_{1,5}$	$a_{2,5}$	$a_{3,5}$

Fig. 2 Transpose matrix

For most of high level programming languages (such as. C), the allocation of elements in the array is row-major. In other words, array elements are placed in the memory with a sequential order of row by row. If the CPU manipulates data in another fashion of column by column, the column elements (such as a_{00}, a_{10}, a_{20}, a_{30}) are not in the consecutive location. The access time for nonconsecutive data is always much slower than the consecutive data. Figure 3 shows the architecture of classical DRAM. To access data in a location, the row address should be decoded before the column address. Figure 4 and Figure 5 show the difference of time required between consecutive and nonconsecutive byte-access. For consecutive-byte access, the row address is decoded only one time, followed by four- column address. while the row address should be decoded for each nonconsecutive-byte. Because of this drawback, our algorithm will arrange data access in a fashion of row by row.

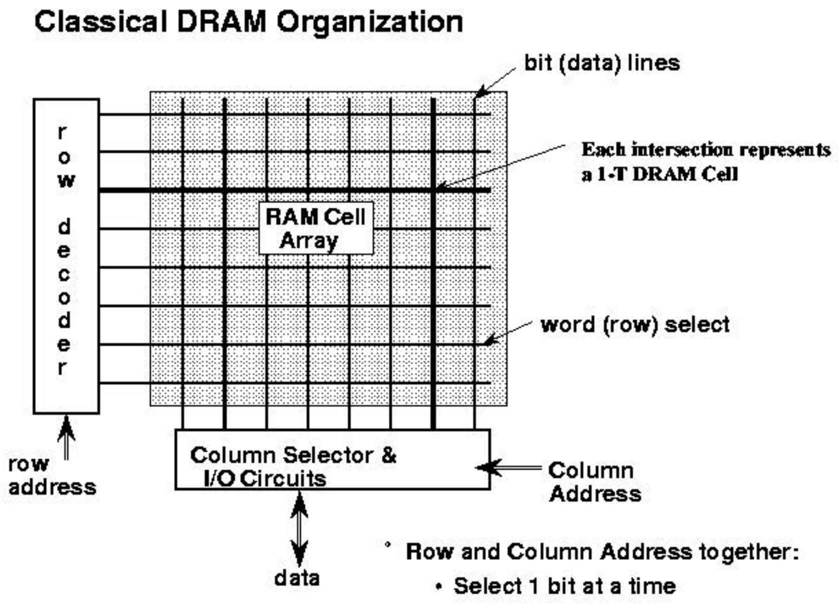
Classical DRAM Organization
bit (data) lines
Each intersection represents a 1-T DRAM Cell
r o w d e c o d e r
RAM Cell Array
word (row) select
row address
Column Selector & I/O Circuits
Column Address
data
Row and Column Address together:
• Select 1 bit at a time

Fig. 3 Classical DRAM architecture

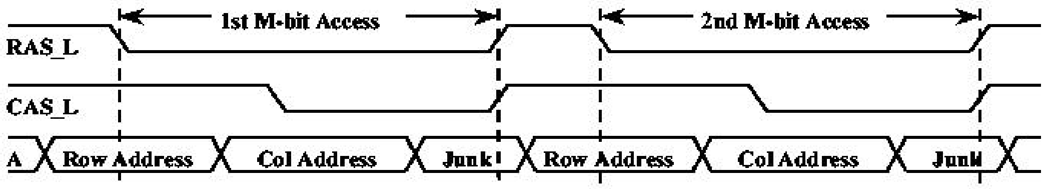

Fig. 4 Nonconsecutive bytes access

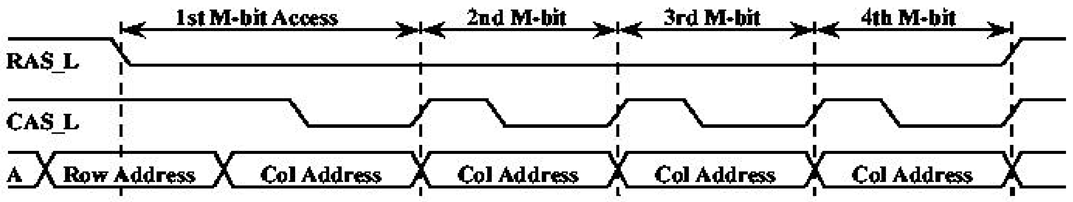

Fig. 5 Consecutive bytes access

1.3 Using word (4 bytes)-manipulation in Transposed Matrix

The original matrix suffers the efficiency problem as described in section 1.2, hence the modified algorithm will always use the transpose matrix. This can also benefit the word manipulation in the row since there is always 4 bytes in each row on the transposed matrix. The original state matrix and cipher key matrix are not uniform in the row. In the original matrix, there are 4, 6, and 8 elements in a row each for 128, 192 and 256 bit data, hence we can not handle the row by word manipulation for all cases.

In the original matrix, there is always four elements in a column for all cases (128, 192 and 256 bits data). After transposing, the new matrix has always 4 elements in the row. Instead of 4 continuous byte-access in the row, our algorithm manipulates it as one word-access. For most of modern computers with 32-bit (or above) CPU, it takes only one access for one word in the row. In C language, it can treat the word (four-byte) data as the type of long integer. The word-manipulation (such as additions) can be performed in this data of long integer type. It will take less than one-fourth of time compared with the byte-manipulation.

1.4. Some ambiguity in the AES proposal

In the section 4.1 of AES proposal, there are some ambiguous descriptions for the state matrix and key matrix. First, we list these paragraphs as follows:

*The input and output used by Rijndael at its external interface are considered to be one-dimensional arrays of 8-bit bytes numbered upwards from 0 to the 4*Nb-1. These blocks hence have lengths of 16,24 or 32 bytes and array indices in the ranges 0..15, 0..23 or0..31. The cipher key is considered to be one-dimensional arrays of 8-bit bytes numbered upwards from 0 to the 4*Nk-1. These blocks hence have lengths of 16,24 0r 32 bytes and array indices in the ranges 0..15, 0..23 or0..31.*

The cipher input bytes are mapped onto the state bytes in the order $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$,$a_{1,1}$,$a_{2,1}$,$a_{3,1}$, …, and The cipher key input bytes are mapped onto the state bytes in the order $k_{0,0}$,$k_{1,0}$,$k_{2,0}$,$k_{3,0}$, $k_{0,1}$,$k_{1,1}$,$k_{2,1}$,$k_{3,1}$, …. At the end of the cipher operation, the cipher output is extracted from the state by taking the state bytes in the same order.

However, there is inconsistent description in another paragraph as follows:

In some instances, these blocks are also considered as one-dimensional arrays of 4-byte vectors, where each vector consists of the corresponding column in the rectangular array representation. These arrays hence have lengths of 4,6 or 8 respectively and indices in the ranges 0..3,0..5 or 0..7. 4-byte vectors will sometimes be referred as words.

The matrix blocks can be considered as one-dimensional arrays of 4-byte (words) vectors, where each vector consists of the corresponding column. These arrays hence have indices in the ranges 0..3, 0..5, 0..7 for 128,192,256 bits block respectively. For C programming language, the column vectors ($[a_{00}\ \ a_{10}\ \ a_{20}\ \ a_{30}]^t$, $[a_{01}\ \ a_{11}\ \ a_{21}\ \ a_{31}]^t$, etc.…) are never allocated in the consecutive location. These vectors can not be directly treated as words. Except that there are other temporary variables or arrays used for storing the transposed column vectors. The transposed vectors are original from the column vectors. The transposed vectors will be $[a_{00}\ \ a_{10}\ \ a_{20}\ \ a_{30}]$, $[a_{01}\ \ a_{11}\ \ a_{21}\ \ a_{31}]$, etc…. This additional procedure needs to be handled for the implementation in row-major programming languages such as C. There will be another overhead in transposing vectors and additional variables or arrays. It will be better that the AES standard should provide another technique note to explain this additional procedure for programmers.

Because of this drawback, we suggest the AES proposal adopt the transposed matrices as indicated in Fig 2. In addition to benefit the efficiency of data access (as explained in section 1.2), it will also resolve the ambiguity problem. If the transposed matrices are adopted in the AES proposal, no additional transposing procedure is needed.

1.5 Related modifications in the Rijndael Algorithm

If the matrix is transposed, the row vectors and column vectors are actually interchanged.
Hence, some modifications should be needed for the Rijndael Algorithm. First, Shiftrow should be modified as Shiftcolumn, Second, Mixcolumn should be modified as Mixrow. After this modification, the Rijndael Algorithm will be clearly understood by both the system designer and the programmer.

2. Conclusion

The Rijndael Algorithm will soon become the standard of AES in summer 2001. Since most of the computer languages are row-major in the matrices, we suggest that NIST should make a transposition to the original state matrix and key matrix to improve the efficiency for the Rijndael Algorithm. If the final standard does not adopt transposed matrix for some reasons, we suggest the AES standard should provide another technique note to explain the ambiguity problem in the AES proposal Section 4.1(as our description in section 1.4). No matter what the final AES standard will be, we suggest programmers should use the transposed matrix in order to achieve better efficiency.

20. Paul Kocher *(05/22/01)*

The proposed algorithm for AES (Rijndael) is an excellent choice, but support for operation with a larger number of rounds is needed.

The round counts in the draft AES FIPS have been chosen to meet aggressive performance targets. While all evidence suggests that the current proposal is adequate to resist current cryptanalytic methods, the safety margin against possible new attacks is relatively small.

This is of serious concern to designers of systems that are expensive to upgrade or must provide extremely long-term security. For example, it is estimated that the cost to upgrade the existing banking infrastructure from DES/3DES to AES would be several billion dollars. For such systems, long-term durability of the algorithm is vastly more important than raw performance, particularly because Moore's Law will compensate for any performance limits.

The round count issue is of the greatest significance for the 256-bit variant (AES-256). Ideally, the 256-bit variant should be 2^128 times harder to break than the 128-bit form. While this is true with respect to exhaustive search, the 256-bit cipher has only 4 extra rounds. Even with these additional rounds, the proposed AES-256 not necessarily more resistant to cryptanalytic attack than AES-128, since the 256-bit cipher is more susceptible to some attacks such as related key attacks. Historically, the greatest error made by cipher designers has been to assume incorrectly that exhaustive search is the easiest attack against an algorithm, and it is important to make sure that this error is not made with AES.

Experience has shown that block ciphers with more rounds are more resistant to cryptanalysis. As a result, I am requesting that NIST standardize an operational mode with a greatly increased round count (such as 128 rounds) for use in conservative designs and applications where performance is of minimal concern. In terms of a specific recommendations, any of the following would provide a suitable conservative mode:

(a) Change the round count for AES-256 from 14 to 128.

(b) Add an additional mode "AES-256B" that is identical to the existing AES-256 except that Nr is 128.

(c) Add an operational mode with 256-bit blocks (Nb=8), 256-bit keys (Nk=8), and 128 rounds (Nr=128).

Thank you for your consideration.

Paul Kocher
President & Chief Scientist
Cryptography Research, Inc.

Paul Kocher — President, Cryptography Research, Inc.
Tel: 415-397-0123 (FAX: -0127) — 607 Market St., 5th Floor
E-mail: paul@cryptography.com — San Francisco, CA 94105

21. Dino Eliades *(5/22/01)*

My only comment for the draft is that as you have a step by step example of the 128-bit round key, it wound be nice to have an example of the round key process when 192 and 256 bit cipher keys are used in "Appendix C - Cipher Example."

Sincerely,
Dino Eliades

22. Carol Widmayer *(5/22/01)*

The comments from the Department of the Treasury are:

1. Announcement, paragraph 6. Interpretation of this paragraph is that agencies may use AES, triple DES, DES, and SKIPJACK for encryption even though DES has been cracked). Is this a correct interpretation?

2. The announcement references FIPS 140-2. However, the publication list indicates that this document is still draft. Will it be published prior to or at the same time as this standard?

3. Web page URL's change frequently as web pages are redesigned. The references to web pages in this document may be the first items to be out-of-date.

4. Section 1. There are three key lengths defined. Will there be guidance to Federal agencies on key lengths and when to use them?

Please contact me should you have any questions.

Carol Ann Widmayer
Department of the Treasury
CIO
Office of Information Systems Security
(202) 622-1110

23. William Harbison *(5/23/01)*

```
<x-html><html>
<font face="Courier New, Courier">AES Draft FIPS Comment:<br> <br>
The original Rijndael AES proposal (Daemen &amp; Rijmen version 2,
3/09/99)<br>
provided extensions for cypher block sizes between 128 and 256 bits
with increments of 32 bits.&nbsp; The Draft FIPS for AES currently
circulating for comments, however, has only adopted the 128-bit block
cypher and apparently offers no provisions for the other block
sizes.&nbsp; To maximize the utility of the AES, it may be beneficial
for the algorithm to support cypher block sizes other than 128
bits.&nbsp; This will be especially true when AES is employed to
provide cover for orderwire communications.&nbsp; Orderwire
communications typically are comprised of short messages (often less
than 200 bits) to minimize overhead.&nbsp; If they are required to be
padded out to meet the 128-bit fixed block size the overhead will be
increased substantially.&nbsp; This may negate the utility of AES for
these types of applications.&nbsp; <br>
<br>
As an example, consider SATCOM applications that utilize orderwire
message signalling for earth terminal control and status.&nbsp; For
these applications, bandwidth efficiency is critical and a restrictive
cipher block size requires a large number of fill (or pad) bits to be
appended to each message.&nbsp; A specific example of this type of
application is the SHF SATCOM Interoperability and Performance Standard
for SHF SATCOM Link Control (MIL-STD-188-166) being developed for the
US Army (CECOM).&nbsp; In this standard, relatively small orderwire
messages (hundreds of bits) are transmitted over relatively low data
rate satellite links (1.2 to 76.8 kbps).&nbsp; With a fixed cypher
block size of 128 bits, the maximum message pad is 127 bits, which in
some cases can be a large percentage of the overall message
length.&nbsp; Alternatively, with a cypher block size that has
increments of 32 bits, the maximum message pad would be 31 bits,
resulting in a significantly more efficient network.&nbsp; To support
this claim, the current MIL-STD-188-166 has a return orderwire channel
operating at 76.8 kbps with 64 TDMA slots supporting 160-bit messages
from 64 earth terminals.&nbsp; In order for this network to conform to
the fixed 128-bit AES cypher block, the orderwire messages would have
to be padded out to 256 bits, which would reduce the number of TDMA
time slots and supported earth terminals to 51.&nbsp; Therefore, for
applications similar to this, it is requested that the FIPS for AES
adopt the cypher block size extensions offered by the original Rijndael
proposal.<br><br>
</font><br>
<div>*******************************************</div>
<div>William Harbison</div>
<div>Linkabit, A Titan Systems Company</div>
<div>Phone: (858) 597-9102</div>
<div>Fax: (858) 552-9660</div>
<br>
*******************************************
</html>
</x-html>
```

24. Jakob Jonsson and Matt Robshaw *(5/29/01)*

Dear NIST AES team,

We would like to congratulate NIST on what appears to be a clear description of the Rijndael cipher.

However we would like to take this opportunity to raise some specific issues about the choice of some of the parameters for the Rijndael cipher and its support within the proposed FIPS document.

First, we are somewhat disappointed to see that only three key sizes are covered within the FIPS document and further, that only one block size is considered. One of the attractive features of Rijndael is the range and flexibility to the user in how these important block cipher parameters are chosen. It seems, however, that much of this flexibility will be lost in a FIPS-compliant implementation of the AES. This lack of support for larger block sizes, such as the 256-bit block size, seems unfortunate when the cipher was designed to explicitly support such choices, and when there is a clear use for such block sizes in modes of use such as hashing.

Second, we would like to echo the comments of many contributors to the AES process, and to observe that the number of rounds in Rijndael appears to allow little margin for security against future developments. It might be observed that the design of the cipher is new to many cryptanalysts, and while the cipher has been elegantly optimized against differential and linear cryptanalysis, there remains considerable mathematical structure within the cipher, the effect of which is unknown and as yet unexplored. While Rijndael was a very popular choice for the AES, there were many who coupled this support with an additional recommendation that more rounds be added to the cipher. We would also like to make this recommendation. Specifically, we suggest adding two extra rounds for each of the three key sizes.

Finally, we would like to point out a minor misprint:

Section 4.2.1, heading:
Replace "Multiplication by y" with "Multiplication by x"

Yours faithfully,

RSA Laboratories

Jakob Jonsson (jjonsson@rsasecurity.com)
Matt Robshaw (mrobshaw@supanet.com)

25. K.Y. Chen et. Al. *(5/29/01)*

Speeding up AES with the Modification of ShiftRow Table

K.Y. Chen, P.D. Chen and C.S. Laih
Department of Electrical Engineering, National Cheng Kung University, Tainan
TAIWAN, ROC.
{peder ccbruce}@crypto.ee.ncku.edu.tw laihcs@eembox.ee.ncku.edu.tw
TEL 2-886-6-2757575-62302
FAX 2-886-6-2761204

Abstract
Recently, NIST has announced that Rijndael has selected to be the Advanced Encryption Standard (AES) and it will be used to replace DES for the following 30 years. In this paper, we proposed a novel method to speed up the implementation of AES with block length 128 bits (Nb=4). It is shown that the proposed method is faster than the original method published in NIST web site about 15.6% with the memory cost of 256KB. The proposed method cannot be applied to block length 192 and 256 bits (Nb=6 and Nb=8) directly. However, with a slight modification on ShiftRow table. Our scheme can be applied to both the block length 192 and 256 bits without decreasing its security. Thus, we suggest NIST adjust the original ShiftRow table to the table we suggest for the consideration of performance.

I. Introduction

The AES standard, Rijnadel Algorithm, was proposed on Oct. 2 2000 by NIST. This algorithm is a symmetry block cipher with 128, 192 and 256-bit key and block length. In [1,2], there are several implemented techniques for speeding up the performance of encryption/decryption. There are some test result tables about AES algorithm in [3]. In this paper, we proposed a novel method to speed up the implementation of AES with block length 128 bits. It is shown that the proposed method can faster than the original method published in NIST web site about 15.6% with the memory of 256KB. The proposed method cannot be applied to block length 192 and 256 bits directly. However, with a slight modification on ROW-SHIFT table. Our scheme can be applied to both the block length 192 and 256 without decreasing its security and the performance. Thus, we suggest NIST should be adjust the original ROW-SHIFT table to the table we suggest due to the performance consideration.

This paper is organized by four sections. Section 2 described the speeding modification and the security of proposed algorithm. The enhanced modification is interpreted in Section 3. Conclusion is finally summarized in section 4.

II. The new algorithm

2.1 The original AES implementation

In order to implement the Rijnadel Algorithm, the AES proposal [1] has the following steps:

Calculating the S-Box values by a lookup table, and this processed step is interpreted by Eq. 1. We assume that the readers are familiar with AES algorithm and the following parameters are the same as of that in [1], unless denoted otherwise.

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = S[p_{0,j}] \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus S[p_{1,j-C_1}] \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S[p_{2,j-C_2}] \begin{bmatrix} 01 \\ 02 \\ 02 \\ 01 \end{bmatrix} \oplus S[p_{3,j-C_3}] \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

To speed up, we have to build a 4KB table that is presented in Eq.2 for getting more efficient enciphering.

$$\begin{gathered} L_0[p] = \begin{bmatrix} S[p] \bullet 02 \\ S[p] \\ S[p] \\ S[p] \bullet 03 \end{bmatrix}, L_1[p] = \begin{bmatrix} S[p] \bullet 03 \\ S[p] \bullet 02 \\ S[p] \\ S[p] \end{bmatrix} \\ L_2[p] = \begin{bmatrix} S[p] \\ S[p] \bullet 03 \\ S[p] \bullet 02 \\ S[p] \end{bmatrix}, L_3[p] = \begin{bmatrix} S[p] \\ S[p] \bullet \\ S[p] \bullet 03 \\ S[p] \bullet 02 \end{bmatrix} \end{gathered} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2)$$

If we have the L_0, L_1, L_2 and L_3 table, then we should do the enciphering step through the following equation.

$$e_j = L_0[p_{0,j}] \oplus L_1[p_{1,j-C_1}] \oplus L_2[p_{2,j-C_2}] \oplus L_3[p_{3,j-C_3}] \quad \ldots\ldots\ldots\ldots\ldots\ldots(3)$$

2.2 Our Implementation

To interpret easily, we describe the data arranged by Table 1(a), the number in Table 1 shows the order of the data store in the memory per block. Due to the communicated law of 'XOR' operation, we can switch the row 2 and row 4 without affecting the processing result. The result is shown in Table 1(b). Since the continuous bytes are arranged (shown by the gray color), we can build up a new lookup table for pre-calculated these two continuous bytes that the processed steps are shown in Eq.s (4) , (5), and (6).

Table 1. the table for speeding up by switching the second and fourth row.

(a). Original

0	4	8	12
5	9	13	1
10	14	2	6
15	3	7	11

(b). After switching

0	4	8	12
15	3	7	11
10	14	2	6
5	9	13	1

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = S\left[p_{0,j}\right]\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus S\left[p_{3,j-C_3}\right]\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \oplus S\left[p_{1,j-C_1}\right]\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S\left[p_{2,j-C_2}\right]\begin{bmatrix} 01 \\ 02 \\ 02 \\ 01 \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \quad \dots\dots\dots\dots(4)$$

or

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} S\left[p_{0,j-C_3}\right]\bullet 02 \oplus S\left[p_{3,j-C_3}\right]\bullet 01 \\ S\left[p_{0,j-C_3}\right]\bullet 01 \oplus S\left[p_{3,j-C_3}\right]\bullet 01 \\ S\left[p_{0,j-C_3}\right]\bullet 01 \oplus S\left[p_{3,j-C_3}\right]\bullet 03 \\ S\left[p_{0,j-C_3}\right]\bullet 03 \oplus S\left[p_{3,j-C_3}\right]\bullet 02 \end{bmatrix} \oplus S\left[p_{1,j-C_1}\right]\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S\left[p_{2,j-C_2}\right]\begin{bmatrix} 01 \\ 02 \\ 02 \\ 01 \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \quad \dots\dots\dots\dots(5)$$

If we let

$$L'_0\left[p_0\, p_1\right] = \begin{bmatrix} S[p_0]\bullet 02 \oplus S[p_1]\bullet 01 \\ S[p_0]\bullet 01 \oplus S[p_1]\bullet 01 \\ S[p_0]\bullet 01 \oplus S[p_1]\bullet 03 \\ S[p_0]\bullet 03 \oplus S[p_1]\bullet 02 \end{bmatrix} \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots(6)$$

then the result of calculating cipher is shown by following equation.

$$e_j = L_0\left[p_{0,j}\right] \oplus L_1\left[p_{1,j-C_1}\right] \oplus L_2\left[p_{2,j-C_2}\right] \oplus L_3\left[p_{3,j-C_3}\right] \oplus k_j \quad \dots\dots\dots\dots(7)$$

2.3 Performance analysis

If we consider a basic operation (XOR operation or a table-lookup) as a CPU clock, then 16 table-lookup and 16 XOR operations will waste 32 CPU clocks. In our proposed scheme, we just need 13 table-lookup and 14 XOR operations in Eq.(7). In the 128-bit block case, we reduce the 5 CPU clocks per enciphering operation. Judging from this viewpoint, the operation increases 15.6% performance. Note that the output result in Eq.(7) is the same as that in Eq.(3).

III. Modification of ShiftRow table

3.1 Discussion in the case of block length 192 and 256

In the Nb=4 condition, we have the continuous bytes for speeding up the Rijndael encipher calculating. But in the Nb=6 and Nb=8 condition, the continuous bytes have disappeared so we cannot get more improvable performance. To solve this problem, our solution is shown as follows:

The Table 2 shows the original ShiftRow parameters. We suggest the modified table be shown in Table 3. The difference between these two tables is the change of the parameter C3. The results of the shifting row operation are shown in Table 4, 5. In this case, we have got 20.8% and 21.8% improvement separately.

Table 2. The shift row parameter

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

Table 3. The modified shift row parameter

Nb	C1	C2	C3
4	1	2	3
6	1	2	5
8	1	3	7

Table 4. The result of Nb=6 rows that shifted by Table 3.

(a). The original elements.

D_{0}	D_{4}	D_{8}	D_{12}	D_{16}	D_{20}
D_{1}	D_{5}	D_{9}	D_{13}	D_{17}	D_{21}
D_{2}	D_{6}	D_{10}	D_{14}	D_{18}	D_{22}
D_{3}	D_{7}	D_{11}	D_{15}	D_{19}	D_{23}

(b). The elements after row shift.

D_{0}	D_{4}	D_{8}	D_{12}	D_{16}	D_{20}
D_{5}	D_{9}	D_{13}	D_{17}	D_{21}	D_{1}
D_{10}	D_{14}	D_{18}	D_{22}	D_{2}	D_{6}
D_{23}	D_{3}	D_{7}	D_{11}	D_{15}	D_{19}

(c). The elements after row communicate.

D_{0}	D_{4}	D_{8}	D_{12}	D_{16}	D_{20}
D_{23}	D_{3}	D_{7}	D_{11}	D_{15}	D_{19}
D_{5}	D_{9}	D_{13}	D_{17}	D_{21}	D_{1}
D_{10}	D_{14}	D_{18}	D_{22}	D_{2}	D_{6}

Table 5. The result of Nb=8 rows that is shifted by Table 3.

(a). The original elements.

D_{0}	D_{4}	D_{8}	D_{12}	D_{16}	D_{20}	D_{24}	D_{28}
D_{1}	D_{5}	D_{9}	D_{13}	D_{17}	D_{21}	D_{25}	D_{29}
D_{2}	D_{6}	D_{10}	D_{14}	D_{18}	D_{22}	D_{26}	D_{30}
D_{3}	D_{7}	D_{11}	D_{15}	D_{19}	D_{23}	D_{27}	D_{31}

(b). The elements after row shift.

D_{0}	D_{4}	D_{8}	D_{12}	D_{16}	D_{20}	D_{24}	D_{28}
D_{5}	D_{9}	D_{13}	D_{17}	D_{21}	D_{25}	D_{29}	D_{1}
D_{10}	D_{14}	D_{18}	D_{22}	D_{26}	D_{30}	D_{2}	D_{6}
D_{31}	D_{3}	D_{7}	D_{11}	D_{15}	D_{19}	D_{23}	D_{27}

(c). The elements after row communicate.

D_{0}	D_{4}	D_{8}	D_{12}	D_{16}	D_{20}	D_{24}	D_{28}
D_{31}	D_{3}	D_{7}	D_{11}	D_{15}	D_{19}	D_{23}	D_{27}
D_{5}	D_{9}	D_{13}	D_{17}	D_{21}	D_{25}	D_{29}	D_{1}
D_{10}	D_{14}	D_{18}	D_{22}	D_{26}	D_{30}	D_{2}	D_{6}

3.2 Security analysis

To analyze the security of our scheme, we define a "diffusion set" to observe diffusion. The set is defined as follows:

$$S_l^r \cong \{C_{l1}^r, C_{l2}^r, .., C_{\ln}^r\}$$

Where C_l^r is recursive diffusion set, *r* is the round number and *l* is row number. And the recursive diffusion set is defined as follows:

$$C_{li}^r = \begin{cases} \{D_{l1}, D_{l2}, ..., D_{\ln}\} & r = 0 \\ \{C_{l1}^{r-1}, C_{l2}^{r-1}, ..., C_{\ln}^{r-1}\} & r > 0 \end{cases}$$

If we have all elements in S_l^r, then we can say S_l^r has the full diffusive property.

For Example: In Nb=4 case, we observe on the column 2 and row 1's element, and we have view on the round 1, then we get a confusion set (by looking up in table 1(a)) $C_4^1 = \{D_4, D_9, D_{14}, D_3\}$ that has four confusion elements D_4, D_9, D_{14}, D_3. In the round 2, the confusion set

$$\begin{aligned} C_4^2 &= \{C_4^1, C_9^1, C_{14}^1, C_3^1\} \\ &= \{\{D_4, D_9, D_{14}, D_3\}, \{D_8, D_{13}, D_2, D_7\}, \{D_{12}, D_1, D_6, D_{11}\}, \{D_0, D_5, D_{10}, D_{15}\}\} \\ &= \{D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, D_{10}, D_{11}, D_{12}, D_{13}, D_{14}, D_{15}\} \end{aligned}$$

done the full diffusion that it mixed by the total 16 data elements. So we can call the Nb=4 case cause the full diffusion in Round 2. Similarly, in original AES case of Nb=6 and Nb=8, *r* is equals to 3 and 4 respectively. In our scheme, the full confusion round number is also the same as the AES done.

3.3 8-bit and 16-bit processor implementation

In our proposed scheme, it slightly communicates two rows for speed up the processing data performance. This modification does not affect the original encrypted processes. Then using the previous methods, the encrypting data step also does not increase any computing power.

IV. Conclusion

In this paper, we suggest a simply row shift method for Rijndael algorithm. With a slight modification on ShiftRow table, our scheme can be applied to both the block length 192 and 256 without decreasing its security and the performance. We get a performance improved by 15.6%. help of 256KB memory. The proposed method is used when AES is implemented in 16-bit processor or about. Besides it does not decrease its security and the performance in 8-bit processor.

Reference

[1] Advanced Encryption Standard proposal, http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf

[2] Advanced Encryption Standard, http://csrc.nist.gov/encryption/aes/

[3] Lawrence E. Bassham III, *Efficiency Testing of ANSI C Implementations of Round 2 Candidate Algorithms for the Advanced Encryption Standard*, The Third Advanced Encryption Standard Candidate Conference, April 2000.

26. Alan Chickinsky *(5/29/01)*

Sir:

In the "Draft Federal Information Processing Standard (FIPS) for the Advanced Encryption Standard (AES)" the data contained in Figure 8 appears to be inconsistant with equations 5.1 and 5.2. Below is the proof showing one inconsistancy.

Figure 8 shows the S-Box translation for Hexadecimal CF to be Hexadecimal 8A. Using equations 5.1 and 5.2 the S-Box translation for Hexadecimal CF results in Hexadecimal 8E not the Hexadecimal 8A in Figure 8.

In the following equations, the symbol "x" stands for the logical XOR function.

Given -

```
B7 = 1
B6 = 1
B5 = 0
B4 = 0
B3 = 1
B2 = 1
B1 = 1
B0 = 1
```

Using equation 5.2 we get-

```
B7' = B7xB6xB5xB4xB3x0 = 1x1x0x0x1x0 = 1
B6' = B6xB5xB4xB3xB2x1 = 1x0x0x1x1x1 = 0
B5' = B5xB4xB3xB2xB1x1 = 0x0x1x1x1x1 = 0
B4' = B4xB3xB2xB1xB0x0 = 0x1x1x1x1x0 = 0
B3' = B7xB3xB2xB1xB0x0 = 1x1x1x1x1x0 = 1
B2' = B7xB6xB2xB1xB0x0 = 1x1x1x1x1x0 = 1
B1' = B7xB6xB5xB1xB0x1 = 1x1x0x1x1x1 = 1
B0' = B7xB6xB5xB4xB0x1 = 1x1x0x0x1x1 = 0
```

which generates an 8E.

The difference between 8E and 8A is an inverted bit 2.

Using equation 5.1, for B2' we get

```
B2' = B2 x B[(2+4) mod8] x B[(2+5) mod8] x B[(2+6) mod8] x B[(2+7)
mod8] x
C2

      = B2 x B6 x B7 x B0 x B1 x C2

      = B7xB6xB2xB1xB0xC2
```

since C2 = 0,

B2' = B7xB6xB2xB1xB0x0 = 1x1x1x1x1x0 = 1

Which is identical to the values found in the equations generated above from
equation 5.2.

Therefore I conclude that the data in the S-Box is incorrect.

Alan Chickinsky
Litton/Tasc
703-633-8300 x 8554