ISSN 0354-9852
Originalni naučni rad
Original scientific paper

UDK 519.857:004.021
DOI 10.7251/AKT1636001D
COBISS.RS-ID 6277656

ALGORITHM AES – STRUCTURE, TRANSFORMATIONS AND PERFORMANCE

Boris Damjanović[1]

Abstract

Today's cryptographic algorithms are designed in a way that they combine mathematical theory and practice of computer science in order to improve resistance to cryptanalysis. Cryptographic algorithms are designed around the binary data format keeping in mind the presumption of hardening possibility of cracking the algorithm. One of the algorithms whose resistance to cryptanalysis during the past 16 years is extensively tested algorithm AES. The Advanced Encryption Standard (AES) is the first cryptographic standard aroused as the result of public competition established by U.S. National Institute of Standards and Technology (NIST). AES has emerged as restriction on winner of this competition, called Rijndael algorithm on the block size of 128 bits. From the moment of its acceptance of the standard in 2001, testing and research of its resistance on cryptanalysis and research focused on improving its performance are made. This paper presents a detailed overview of the algorithm AES, together with all its transformations and with ideas to speed up its work.

Keywords: *Cryptogaphy, Algorithm AES, performance*
JEL classificaiton:C00

INTRODUCTION

Today's cryptographic algorithms are designed in a way that they combine complex mathematical procedures and the theory and practice of computer science in order to improve resistance to cryptanalysis. As stated in [1] AES (Advanced Encryption Standard) algorithm is a complex cipher whose resistance to cryptanalysis has been extensively tested over the last 15 years. This algorithm has become the de facto global standard for commercial and open source software and hardware. In addition to the U.S. administration, a number of institutions and individuals around the world use this algorithm. Advanced Encryption Standard (AES) is the first cryptographic standard aroused as a result of public competition that was esta-

1 Boris Damjanović Ph.D., Banja Luka College, Banja Luka; boris.damanovic@blc.edu.ba

blished by U.S. National Institute of Standards and Technology. Standard can theoretically be divided into three cryptographic algorithms: AES-128, AES-192 and AES-256.

In January 1997, the US National Institute of Standards and Technology (NIST) announced the start of an initiative to develop a new encryption standard: the AES . The new encryption standard was to become a Federal Information Processing Standard (FIPS), replacing the old Data Encryption Standard (DES) and triple-DES. Unlike the selection of prior cryptographic algorithms, NIST had announced that the AES selection process would be open. Anyone could submit a candidate cipher. Each submission, provided it met the requirements, would be considered on its merits. NIST would not perform any security or efficiency evaluation itself, but instead invited the cryptology community to mount attacks and try to crypt analyse the different candidates, and anyone who was interested to evaluate implementation cost. All results could be sent to NIST as public comments for publication on the NIST AES web site or be submitted for presentation at AES conferences. NIST would merely collect contributions using them to base their selection. NIST would motivate their choices in evaluation reports [2].

NIST has prescribed the following rules [3]:

- AES shall be publicly defined.
- AES shall be a symmetric block cipher.
- AES shall be designed so that its key length may be increased as needed.
- AES shall be implementable in both hardware and software.
- AES shall either be
  - freely available, or
  - available under terms consistent with the ANSI Patent Policy.
- Algorithms which meet the above requirements will be judged based on the following factors:
  - security (resistance to cryptanalysis),
  - computational efficiency,
  - memory requirements,
  - hardware and software suitability,
  - simplicity,
  - flexibility, and
  - licensing requirements

The required effort to produce a 'complete and proper' submission package would already filter out several of the proposals. The 15 submissions that were completed in time and accepted were: CAST-256, Crypton, DEAL, DFC, E2, Frog, HPC, LOKI97, Magenta, Mars, RC6, Rijndael, SAFER+, Serpent and Twofish [1, 2, 4, 5, 6].

After the series of workshops (Ventura-California, august 1998., Rome, march 1999.) and researchs conducted and papers published, there was a relatively calm period that ended with the announcement of the five candidates by NIST in August 1999. The finalists were: MARS, RC6, Rijndael, Serpent and Twofish [1, 2, 4, 5, 6].

Next round of testing was held in April 2000 in New York at a conference that was dedicated to this algorithm. The conference was combined with the results of the next workshop titled Fast Software Encryption Workshop, which was also held in New York [1, 2].

On 2 October, 2000, NIST officially announced that Rijndael would become Advanced Encryption Standard [1, 2].

AES algorithm, as defined by FIPS-197 document [7] cites a data block must be always 128 bit-long, while the key sizes could be 128, 192 or 256 bits. On the other hand, Rijndael (from which AES evolved) allows for both key and block sizes to be chosen from the set of {128, 160, 192, 224, 256} bits. The very fact that AES is really just a subset of Rijndael emphasize its large flexibility.

As mentioned, AES algorithm described in FIPS-197 document [7, 8, 9] transforms 128 bit block of data during 10, 12 or 14 rounds using the initial key lengths of 128, 192 and 256 bit. The initial key is then enlarged to (10+1)*16, (12+1)*16 or (14+1)*16 bytes in the key expansion routine. Each round repeats the SubBytes(), MixColumns(), ShiftRows() and AddRoundKey() transformations. AES authors redefine both addition operation within the GF(2^8), which is then conducted by XOR operation at the byte level and multiplication operation which is thus conducted as polynomial multiplication with the conditional modulo polynomial 0x11B. The mentioned multiplication is the most time consuming in the aspect of optimization, because it is intensively used during the MixColumns() transformation.

Inverse cipher transforms 128 bit block of ciphertext during 10, 12 or 14 rounds using the same keys as cipher. Each round repeats the InvSubBytes(), InvMixColumns(), InvShiftRows() and AddRoundKey() transformations.

In the following text, the theoretical assumptions and basic transformation of this algorithm will be displayed.

MATHEMATICAL PRELIMINARIES

The field GF(28)

Every byte of data in the AES algorithm is treated as a series of bits which are elements of a finite field. A byte b, consisting of bits b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0, is considered as a polynomial with coefficient in {0,1} [10]:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_ix^i \quad (1)$$

For example, {01100011} identifies the specific finite field element [7]

$$x_6 + x_5 + x + 1 \quad (2)$$

Addition

Authors of the AES algorithm redefine the addition operation. The addition [7] of two elements in a finite field is achieved by "adding" the coefficients for the corresponding powers in the polynomials for the two elements. The addition is performed with the XOR operation (denoted by $\oplus$) - i.e., modulo 2 - so that

$1 \oplus 1 = 0$, $1 \oplus 0 = 1$, and $0 \oplus 0 = 0$.

For example:

$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$, as polynomial:

$\{01010111\} \oplus \{10000011\} = \{11010100\}$, binary (3)

$\{57\} \oplus \{83\} = \{d4\}$, hexadecimal.

Consequently, subtraction of polynomials is identical to addition of polynomials.

Multiplication

In the polynomial representation, multiplication in GF(2^8) (denoted by $\bullet$) corresponds with the multiplication of polynomials modulo an irreducible polynomial of degree 8. A polynomial is irreducible if its only divisors are one and itself. For the AES algorithm, this irreducible polynomial is

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad (4)$$

or in hexadecimal representation.:

{01}{1b} (5)

For example:

57h $\bullet$ 83h

$\{57\} \bullet \{83\} = \{c1\}$

$1010111 \bullet 10000011 = 11000001$

$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^7 +$ (6)

$x^7 + x^5 + x^3 + x^2 + x +$

$x^6 + x^4 + x^2 + x + 1$

$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

Finally:

$$(x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1) \bmod (x^8 + x^4 + x^3 + x + 1) = x^7 + x^6 + 1 \quad (7)$$

Dividing with this polynomial ensures that the result will always be a binary polynomial of degree less than 8 so that it can be represented by a single byte.

Multiplication by x

As stated in [1, 7], if we have to multiply the following polynomial:

$$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0 \quad (8)$$

With polynomial x, we get the result:

$$b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x^1 \quad (9)$$

The result of multiplication of the x*b(x) is then reduced to the degree 8 by applying the modulo operation with irreducible polynomial m(x). If b7 = 0, the polynomial is already in the reduced form. If b7 = 1, the reduction is carried out by subtracting (using an XOR operation) the polynomial m (x) [1].

At the bit level, multiplication by x ({00000010} or {02}) can be done by using the left Shift after which the (conditional) follows with xor {00011011} or {1b}.

For example, to multiply {57} • {13} = {fe}, we wil use [1]:

$$\begin{array}{ll} \{57\} \bullet \{01\} = \{57\} & 1 \\ \{57\} \bullet \{02\} = \text{xtime}(\{57\}) = \{ae\} & 1 \\ \{57\} \bullet \{04\} = \text{xtime}(\{ae\}) = \{47\} & 0 \\ \{57\} \bullet \{08\} = \text{xtime}(\{47\}) = \{8e\} & 0 \\ \{57\} \bullet \{10\} = \text{xtime}(\{8e\}) = \{07\} & 1 \end{array} \quad (10)$$

Bearing in mind that the {13h} is equal to {10011b}, if we need to multiply {57} • {13} = {fe}, we will use:

$$\begin{aligned} \{57\} \bullet \{13\} &= \{57\} \bullet (\{01\} \oplus \{02\} \oplus \{10\}) \\ &= \{57\} \oplus \{ae\} \oplus \{07\} \\ &= \{fe\} \end{aligned} \quad (11)$$

Alternativelly, description of AES multiplicatio is given in [11], where is stated that the finite field element {00000010} is the polynomial x, which means that multiplying another element by this value increases all it's powers of x by 1. This is equivalent to shifting its byte representation up by one bit so that the bit at position i moves to postion i+1. If the top bit is set prior to this move it will overflow to create an x^8 term, in which case the modular polynomial is added to cancel this additional bit to leave a result that fits within a byte. When {11001000} is multiplied by x, {00000010}, the initial result is 1{10010000}. The 'overflow' bit is then removed by adding the modular polynomial, 1{00011011}, using an exclusive-or operation to give the final result as {10001011}.

Polynomials with coefficients in gf(28)

Four term polynomials can be defined with coefficients that are finite field elements as [7, 11]:

$$a(x) = a_3x^3 + a_2x^2 + a_1x^1 + a_0 \quad (12)$$

where the four coefficients will be denoted as a word in the form [a0 , a1 , a2 , a3] (note that the index increases from left to right in this notation) [11]. With a second polynomial:

$$b(x) = b_3x^3 + b_2x^2 + b_1x^1 + b_0 \quad (13)$$

define a second four-term polynomial.

Multiplication is achieved in two steps. In the first step, the polynomial product c(x) = a(x) • b(x) is algebraically expanded, and like powers are collected to give

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x^1 + c_0 \quad (14)$$

where

$$\begin{aligned} c_0 &= (a_0 \bullet b_0), \\ c_1 &= (a_0 \bullet b_1) \oplus (a_1 \bullet b_0), \\ c_2 &= (a_0 \bullet b_2) \oplus (a_1 \bullet b_1) \oplus (a_2 \bullet b_0), \\ c_3 &= (a_0 \bullet b_3) \oplus (a_1 \bullet b_2) \oplus (a_2 \bullet b_1) \oplus (a_3 \bullet b_0), \\ c_4 &= (a_1 \bullet b_3) \oplus (a_2 \bullet b_2) \oplus (a_3 \bullet b_1), \\ c_5 &= (a_2 \bullet b_3) \oplus (a_3 \bullet b_2), \\ c_6 &= (a_3 \bullet b_3). \end{aligned} \quad (15)$$

The result, c(x), does not represent a four-byte word. Therefore, the second step of the multiplication is to reduce c(x) modulo a polynomial of degree 4; the result can be reduced to a polynomial of degree less than 4. For the AES algorithm [1, 11], this is accomplished with the polynomial

$$x^4 + 1 \quad (16)$$

so that:

$$x^i \bmod (x^4 + 1) = x^{i \bmod 4} \quad (17)$$

The modular product of a(x) and b(x), denoted by a(x) ⊗ b(x), is given by the four-term polynomial d(x), defined as follows [1, 11]:

$$d(x) = d_3x^3 + d_2x^2 + d_1x^1 + d_0 \quad (18)$$

where

$$\begin{aligned}&d(x) = c(x) \bmod (x^4+1) = (c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x^1 + c_0) \bmod (x^4+1) = \\ &c_6x^{6 \bmod 4} + c_5x^{5 \bmod 4} + c_4x^{4 \bmod 4} + c_3x^{3 \bmod 4} + c_2x^{2 \bmod 4} + c_1x^{1 \bmod 4} + c_0x^{0 \bmod 4} = \\ &c_6x^2 + c_5x^1 + c_4x^0 + c_3x^3 + c_2x^2 + c_1x^1 + c_0x^0 = \\ &c_3x^3 + c_6x^2 + c_2x^2 + c_5x^1 + c_1x^1 + c_4x^0 + c_0x^0 = \\ &d_3x^3 + d_2x^2 + d_1x^1 + d_0\end{aligned} \qquad (19)$$

and

$$\begin{aligned}d_0 &= c_0 \oplus c_4 = (a_0 \bullet b_0) \oplus (a_1 \bullet b_3) \oplus (a_2 \bullet b_2) \oplus (a_3 \bullet b_1), \\ d_1 &= c_1 \oplus c_5 = (a_0 \bullet b_1) \oplus (a_1 \bullet b_0) \oplus (a_2 \bullet b_3) \oplus (a_3 \bullet b_2), \\ d_2 &= c_2 \oplus c_6 = (a_0 \bullet b_2) \oplus (a_1 \bullet b_1) \oplus (a_2 \bullet b_0) \oplus (a_3 \bullet b_3), \\ d_3 &= c_3 = (a_0 \bullet b_3) \oplus (a_1 \bullet b_2) \oplus (a_2 \bullet b_1) \oplus (a_3 \bullet b_0).\end{aligned} \qquad (20)$$

When a(x) is a fixed polynomial, the operation defined in equation (18) can be written in matrix form as [1, 7, 11]:

$$\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} \text{б}_0 \\ \text{б}_1 \\ \text{б}_2 \\ \text{б}_3 \end{bmatrix}$$

Figure 1: Modular product a(x) ⊗ b(x)

Because x4 +1 is not an irreducible polynomial over GF(2^8), multiplication by a fixed four-term polynomial is not necessarily invertible. However, the AES algorithm specifies a fixed four-term polynomial that does have an inverse [1, 7, 11]

$$\begin{aligned}a(x) &= \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \\ a^{-1}(x) &= \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}\end{aligned} \qquad (21)$$

Another polynomial used in the AES algorithm (RotWord function) has coefficients [1]:

$a^0a^1a^2 = \{00\}$ i

$a^3 = \{01\}$

which is the polynomial x^3. If this polynomial is inserted in the previous matrix, we have [1]

$$\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 0_0 & 1_3 & 0_2 & 0_1 \\ 0_1 & 0_0 & 1_3 & 0_2 \\ 0_2 & 0_1 & 0_0 & 1_3 \\ 1_3 & 0_2 & 0_1 & 0_0 \end{bmatrix} \begin{bmatrix} \text{б}_0 \\ \text{б}_1 \\ \text{б}_2 \\ \text{б}_3 \end{bmatrix} = \begin{bmatrix} 0*\text{б}_0 + 1*\text{б}_1 + 0*\text{б}_2 + 0*\text{б}_3 \\ 0*\text{б}_0 + 0*\text{б}_1 + 1*\text{б}_2 + 0*\text{б}_3 \\ 0*\text{б}_0 + 0*\text{б}_1 + 0*\text{б}_2 + 1*\text{б}_3 \\ 1*\text{б}_0 + 0*\text{б}_1 + 0*\text{б}_2 + 0*\text{б}_3 \end{bmatrix} = \begin{bmatrix} \text{б}_1 \\ \text{б}_2 \\ \text{б}_3 \\ \text{б}_0 \end{bmatrix}$$

Figure 2: RotWord

which gives the effect of rotation of bytes in word. Polynomial [b0, b1, b2, b3] is transformed into [b1, b2, b3, b0].

Array state

As stated in [11], Rijndael operations are internally performed on a two dimensional array of bytes called the state that consists of 4 rows of bytes, each of which contains Nb bytes, where Nb is the input sequence length divided by 32. In the state array, denoted by the symbol s, each individual byte has two indexes: its row number r, in the range 0 £ r < 4, and its column number c, in the range 0 £ c < Nb, hence allowing it to be referred to either as sr,c or as s[r, c]. For AES the range for c is 0 £ c < 4 since Nb has a fixed value of 4.

At the start (end) of an encryption or decryption operation the bytes of the cipher input (output) are copied to (from) this state array in the order shown in Figure 1.

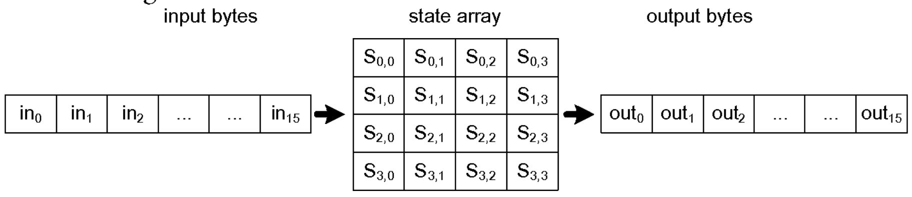
input bytes
state array
output bytes
in_0 in_1 in_2 ... ... in_{15}
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{1,1}$ $S_{1,2}$ $S_{1,3}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$
out_0 out_1 out_2 ... ... out_{15}

Figure 3: Input to the cipher state array and output from it

ALGORITHM SPECIFICATION

Rijndael is a key-iterated block cipher: it consists of the repeated application of a round transformation on the state. The number of rounds is denoted by N r and depends on the block length and the key length. [2].

The cipher transformation

Encryption function encompasses four AES transformations. Pseudo code encryption functions can be represented in the next few lines [7].

```
/* in out keys w */
Cipher (byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
byte state[4, Nb]
state = in
/* initial mixing with key */
AddRoundKey( state, w[0, Nb-1])

/* „common“ rounds */
For round = 1 step 1 to Nr-1
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
End for

/* final round */
SubBytes(state)
ShiftRows(state)
AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

Out = state
End
```

Listing 1: Pseudo code for Cipher() function

As stated in [7], individual transformations SubBytes() ShiftRows() MixColumns(), and AddRoundKey() transform the buffer State during each round. From the pseudo code can be seen that only the final round is different from previous Nr rounds, in that it does not perform the MixColumns() function. The article [2] states that the authors changed the names of some transformations according to the suggestions Dr. B.Gladmana in relation to the original document filed.

Subbytes transformation

The SubBytes() transformation is a non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-box). This invertible S-box is constructed by composing two transformations [1, 7]:

1 .Take the multiplicative inverse in the finite field GF(2^8), the element {00} is mapped to itself.

2. Apply the following affine transformation

$$б'_и = б_и \oplus б_{(и+4)\,мод\,8} \oplus б_{(и+5)\,мод\,8} \oplus б_{(и+6)\,мод\,8} \oplus б_{(и+7)\,мод\,8} \oplus ц_и \qquad (22)$$

for 0 <= i < 8 , where b_i is the i_{th} bit of the byte, and c_i is the ith bit of a byte c with the value {63} or {01100011}.

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Figure 4: Single S-Box element

Effect of the SubBytes() transformation on the State is

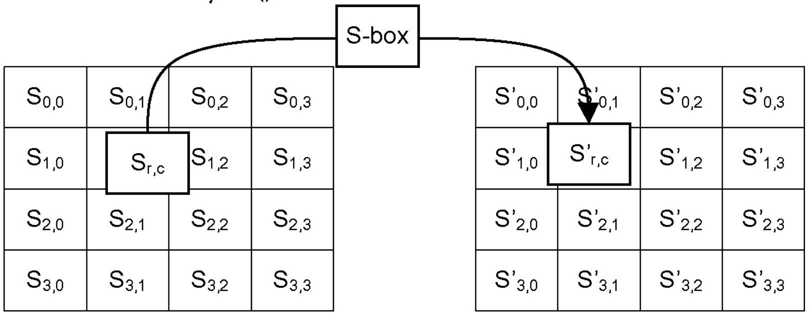

Figure 5: Applying SubBytes on single State element

Shiftrows transformation

As stated in [7], in the ShiftRows() transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, r = 0, is not shifted.

$$S'_{r,c} = S_{r,(c + \mathrm{shift}(r, Nb)) \bmod Nb} \quad \text{za } 0 < r < 4 \text{ i } 0 \le c < Nb, \qquad (23)$$

Figure 6 illustrates the ShiftRows() transformation.

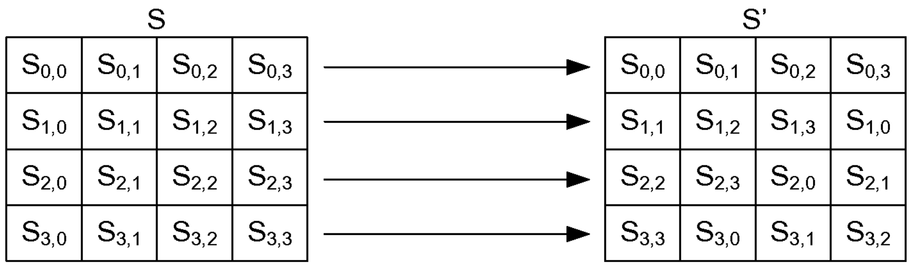

Figure 6: Applying SubBytes on single State element

Mixcolumns transformation

The MixColumns() transformation operates on the State column-by--column, treating each column as a four-term polynomial. The columns are considered as polynomials over GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial a(x), given by [1, 7]

$$a(x) = \{03\}x3 + \{01\}x2 + \{01\}x + \{02\} \qquad (24)$$

this can be written as a matrix multiplication. Let

$$s'(x) = a(x) \otimes s(x) \qquad (25)$$

$$\begin{bmatrix} b_{0,c} \\ b_{1,c} \\ b_{2,c} \\ b_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,c} \\ c_{1,c} \\ c_{2,c} \\ c_{3,c} \end{bmatrix}$$

Figure 7: MixColumns transformation as matrix

The four bytes in a column are replaced by the following

$$\begin{aligned} S'_{0,c} &= (\{02\} \bullet s_{0,c}) \otimes (\{03\} \bullet s_{1,c}) \otimes (\{01\} \bullet s_{2,c}) \otimes (\{01\} \bullet s_{3,c}) \\ S'_{1,c} &= (\{01\} \bullet s_{0,c}) \otimes (\{02\} \bullet s_{1,c}) \otimes (\{03\} \bullet s_{2,c}) \otimes (\{01\} \bullet s_{3,c}) \\ S'_{2,c} &= (\{01\} \bullet s_{0,c}) \otimes (\{01\} \bullet s_{1,c}) \otimes (\{02\} \bullet s_{2,c}) \otimes (\{03\} \bullet s_{3,c}) \\ S'_{3,c} &= (\{03\} \bullet s_{0,c}) \otimes (\{01\} \bullet s_{1,c}) \otimes (\{01\} \bullet s_{2,c}) \otimes (\{02\} \bullet s_{3,c}) \end{aligned} \qquad (26)$$

Addroundkey transformation

In this transformation , the state is modified by combining it with a round key with the bitwise XOR operation [2]. Nb words from the key schedule are each added (XOR'd) into the columns of the state so that [1, 7]:

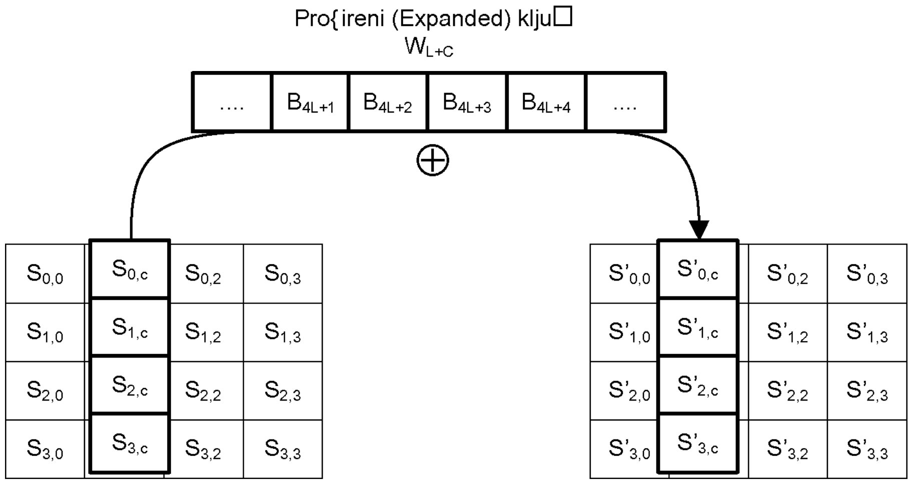

Figure 8: AddRoundKey transformation

The key schedule

As in [11] the round keys are derived from the cipher key by means of a key schedule with each round requiring Nb words of key data with an extra initial set making Nb(Nr + 1) words in total. The resulting key schedule consists of a linear array of 4-byte words, denoted [wi], with i in the range

$0 <= i < Nb(Nr + 1)$.

The function RotWord(x) takes a word $[b_0 , b_1 , b_2 , b_3]$ as input and returns the word

$[b_1 , b_2 , b_3 , b_0]$.

The word array Rcon[i] contains the values given by $[x^{i-1}, 0, 0, 0]$ with x^{i-1} being powers of x in the field GF(256) (note that i starts at 1, not 0).

```
KeyExpansion( byte key[4*Nk], word w[Nb*(Nr + 1)], Nk)
begin
    word temp
    i = 0
    while (i < Nk)
        w[i] = word( key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i + 1
    end while
    i = Nk
    while ( i < Nb*(Nr + 1) )
        temp = w[i-1]
        if ((i mod Nk) = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if ( (Nk > 6) and ((i mod Nk) = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i - Nk] xor temp
        i = i + 1
    end while
end
```

Listing 2: Pseudo code for KeyExpansion

INVERSE CIPHER

AES Decryption computes the original plaintext of an encrypted ciphertext. During the decryption, the AES algorithm reverses encryption by executing inverse round transformations in reverse order. The round transformation of decryption uses the functions AddRoundKey, InvMixColumns, InvShiftRows, and InvSubBytes [12]. The Inverse Cipher is described in following pseudo code [1]:

```
InvCipher (byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
byte state[4, Nb]
state = in
AddRoundKey( state, w[Nr*Nb, (Nr+1)*Nb-1])

For round = Nr-1 step -1 downto 1
InvShiftRows(state)
InvSubBytes(state)
AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
InvMixColumns(state)
End for

InvShiftRows(state)
InvSubBytes(state)
AddRoundKey(state, w[0, Nb-1])

Out = state
End
```

Listing 3: Pseudo code for InvCipher() function

PERFORMANCE IMPROVEMENT

Today's cryptographic algorithms are designed in a way that they combine mathematical theory and practice of computer science in order to improve resistance to cryptanalysis. Complexity of cryptographic algorithms caused the respective requirements in terms of processing power.

In order to increase performance, hardware manufacturers today are using multiprocessor systems or hardware accelerators. The first request requires a special way of code writing from the manufacturers of the system as well as cryptographic software. The second request from the software manufacturer imposes an obligation to adapt to the specific drivers or special instruction sets. A special kind of improvements is represented by the research of potential acceleration of modes of operations by using parallelization.

When it comes to performance of this algorithm, there is the potential use of specific solutions, such as eg. assembler language or C / C ++ for the AES-NI instruction set. There are a number of works that examine the possibility of algorithm AES acceleration [9, 13, 14, 15, 16] on different platforms and programming languages.

Recently, there have been emerged three programming paradigm connected with AES algorithm acceleration. There are a number of studies [17, 18, 19] that have focused on the performance improvement of this algorithm by using parallelization of its execution.

In terms of performance improvements, there are solutions that use the GPU parallelization and performance improvement as [1, 20, 21, 22].

In his study Manavski [22] discusses the possibility of the implementation of the algorithm AES using at that time traditional approach, that is based on the OpenGL library as well as the implementation using the NVIDIA CUDA platform and determine the benefits that are achieved using the new architecture.

The existence of different modes of operations has already served as an idea for parallelization of execution of some algorithms. According to Lipmaa et al. [23] blocks C1, C2, ... can also be encrypted at the same time and therefore CTR mode can be paralelized.

The rapid development of GPU (Graphic Processing Unit) units and user environments such as NVidia CUDA or OpenCL has led to various attempts AES algorithm parallelization using CTR mode. Thus, according to Tran et al. [24] the authors first increase the size of the block in relation to the standards defined by AES algorithm, and then use the coarse grained granularity for algorithm parallelization. Authors in the solution shown in (Di Biagio et al. 2009) using a fine (fine grained) and internal granularity parallelism of each round by independently manipulating with 4 32-bit words (T-word) that occur in each AES-this round. Authors in [21] are using a fine grained granularity and internal parallelism of each round by independently manipulating with 4 32-bit words (T-word) that occur in each AES-this round.

In an Intel document [25] author present the source code of programs that are generated keys and 128-bit, 192-bit and 256-bit encryption and decryption in ECB, CBC and CTR cryptographic modes. This text presents the source code for the simultaneous (parallel) processing of 4 blocks of data in the ECB, and the CTR mode and decryption in CBC mode. All examples can be compiled using Intel C/C ++ compiler v11 or later.

Hoban et al. [26] in exploring the possibility of improving the method for encryption within the operating system Linux provide a solution for the performance of this algorithm parallelization using XTS encryption mode. In [27] authors examine possibilities of achieving performance improvement by using new parallel tweakable OFB modes of operation. In creation of these new modes, XEX and XE constructions and XTS-AES multiplication are used.

As stated in [1], research has shown that the greatest acceleration in the execution of the algorithm can be achieved using aes-ni instruction set. In addition, use of GPU units for AES algorithm parallelization has enormous potential for the performance acceleration of this algorithm. Use of the GPU units in programs have shown almost equal performance with programs using Ni-AES instruction set.

CONCLUSION

In this paper gives an overview of the algorithm AES with its transformations and basic directions and methods used to accelerate its execution. A characteristic of today's conventional computer system is slowing the growth rate of single-processor systems and focusing hardware manufacturers on multi-processor systems and hardware accelerators. In order to increase AES algorithm performance, programmers and hardware manufacturers today are using parallel programming or hardware accelerators. There are great number of researches that have been focused on the performance improvement of this algorithm by using parallelization of its execution. A substantial part of this research is devoted to parallelization using GPU devices, where they achieved very significant results. Exploration of the possibilities for parallelization of individual modes of operation is another direction in which they move numerous studies. However, hardware acceleration, such as AES-NI acceleration is currently unmatched when it comes to speeding up AES algorithm.

Sažetak

Moderna kriptografija se u velikoj mjeri oslanja na kompleksne algoritme koji su zasnovani na matematičkoj teoriji i na praksi računarskih nauka. Današnji kriptografski algoritmi se dizajniraju oko binarnog formata podataka imajući pri tome u vidu i pretpostavku težine izračunavanja da bi što više otežali mogućnost njihovog razbijanja. Jedan od algoritama čija je otpornost na kriptoanalizu tokom prethodnih 16 godina intenzivno testirana je algoritam AES.

Advanced Encryption Standard (AES) je prvi kriptografski algoritam koji je nastao kao rezultat javno objavljenog i održanog takmičenja od strane NIST instituta (National Institute of Standards and Technology) 1997. godine da bi se pronašao algoritam koji će postati slijedeći standard američke vlade. AES je nastao restrikcijom pobjednika ovog takmičenja, algoritma pod nazivom Rijndael, na blok veličine 128 bita. Od momenta njegovog prihvatanja za standard 2001. godine traju neprekidna testiranja i istraživanja njegove otpornosti na kriptoanalizu ali i testiranja i istraživanja skoncentrisana na poboljšanje njegovih performansi. Ovaj rad predstavlja detaljan pregled algoritma AES zajedno sa svim njegovim transformacijama u i sa idejama za ubrzanje njegovog rada.

Ključne riječi: *kriptografija, algoritam AES, performanse*

REFERENCES

1. Damjanović B. (2016) Adaptive implementation of the AES algorithm in modern operating systems, Ph.D. thesis, University of Belgrade, Faculty of Organizational Sciences
2. Daemen J., Rijmen V., (2002). The Design of Rijndael, Springer-Verlag, Inc.
3. KONHEIM A. G., Computer Security And Cryptography, John Wiley & Sons, Inc., Hoboken, New Jersey, 2007
4. Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., & Ferguson, N. (1999). Performance comparison of the AES submissions.
5. Biham, E. (1999, March). A note on comparing the AES candidates. In *Second AES Candidate Conference* (Vol. 266, pp. 22-23).
6. Bassham, L. E. (1999). *Efficiency testing of ANSI C implementations of round 1 candidate algorithms for the advanced encryption standard.* US Department of Commerce, Technology Administration, National Institute of Standards and Technology.
7. Specification for the ADVANCED ENCRYPTION STANDARD (AES), (2001) Federal Information Processing Standards Publication 197, Available at: http://csrc.nist.gov/publications/.
8. Satoh, A., Morioka, S., Takano, K., & Munetoh, S. (2001, December). A compact Rijndael hardware architecture with S-box optimization. In *International Conference on the Theory and Application of Cryptology and Information Security* (pp. 239-254). Springer Berlin Heidelberg.
9. DAMJANOVIĆ, B., & SIMIĆ, D. (2013). Performance evaluation of AES algorithm under Linux operating system. *Proceedings of the Romanian Academy Series A–Mathematics Physics Technical Sciences Information Science, 14*(2).
10. Daemen J., Rijmen V., (1998) The Rijndael Block Cipher Proposal, NIST, Available at: csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf
11. Gladman, B., (2007), A Specification for Rijndael, the AES Algorithm, Available at: http://gladman.plushost.co.uk/oldsite/cryptography_technology/rijnda
12. Hyubgun Lee, Kyounghwa Lee, Yongtae Shin, AES Implementation and Performance Evaluation on 8-bit Microcontrollers, (IJCSIS) International Journal of Computer Science and Information Security, Vol. 6 No. 1, 2009
13. Elbirt, A. J., Yip, W., Chetwynd, B., & Paar, C. (2001). An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 9*(4), 545-557.
14. Mandal, A. K., Parakash, C., & Tiwari, A. (2012, March). Performance evaluation of cryptographic algorithms: DES and AES. In *Electrical, Electronics and Computer Science (SCEECS), 2012 IEEE Students' Conference on* (pp. 1-5). IEEE.
15. Elminaam, D. S. A., Kader, H. M. A., & Hadhoud, M. M. (2008). Performance evaluation of symmetric encryption algorithms. *IJCSNS International Journal of Computer Science and Network Security, 8*(12), 280-286.
16. Singhal, N., & Raina, J. P. S. (2011). Comparative analysis of AES and RC4 algorithms for better utilization. *International Journal of Computer Trends and Technology, 2*(6), 177-181.
17. Saggese, G. P., Mazzeo, A., Mazzocca, N., & Strollo, A. G. (2003, September). An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm. In *International Conference on Field Programmable Logic and Applications* (pp. 292-302). Springer Berlin Heidelberg.
18. Yoo, S. M., Kotturi, D., Pan, D. W., & Blizzard, J. (2005). An AES crypto chip using a high-speed parallel pipelined architecture. *Microprocessors and Microsystems, 29*(7), 317-326.

19. Granado-Criado, J. M., Vega-Rodríguez, M. A., Sánchez-Pérez, J. M., & Gómez-Pulido, J. A. (2010). A new methodology to implement the AES algorithm using partial and dynamic reconfiguration. *INTEGRATION, the VLSI journal, 43*(1), 72-80.
20. Iwai, K., Kurokawa, T. and Nisikawa, N. (2010), AES Encryption Implementation on CUDA GPU and Its Analysis, First International Conference on Networking and Computing (ICNC)
21. Di Biagio, A., Barenghi, A., Agosta and G. and Pelosi, G. (2009), Design of a Parallel AES for Graphics Hardware using the CUDA framework, IEEE International Symposium onParallel & Distributed Processing
22. Manavski, S.A. (2007), CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography, IEEE International Conference on Signal Processing and Communications,
23. Lipmaa H., Rogaway P and Wagner D., (2000), Comments to NIST Concerning AES--modes of Operations: CTR-mode Encryption. In Symmetric Key Block Cipher Modes of Operation Workshop, Baltimore, Maryland, USA
24. Tran, N. P., Lee, M., Hong, S., & Lee, S. J. (2011, August). Parallel execution of AES-CTR algorithm using extended block size. In *Computational Science and Engineering (CSE), 2011 IEEE 14th International Conference on* (pp. 191-198). IEEE.
25. Gueron, S. (2012), Intel Corporation, White Paper, Intel Advanced Encryption Standard (AES) New Instructions Set
26. Hoban, A., Laurent, P., Betts, I. and Tahhan, M. (2013), Unleashing Linux*- Based Secure Storage Performance with Intel AES New Instructions, Intel Corporation, White Paper,
27. Damjanović, B. and Simić, D. (2015), Tweakable parallel OFB mode of operation with delayed thread synchronization, Wiley, Security and Communication Networks, Security Comm. Networks (2015), Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1404.