From AES to Dynamic AES

Pablo Freyre, Oristela Cuellar, Nelson Díaz and Adrián Alfonso

***Abstract*—The cryptographic algorithm AES (Advanced Encryption Standard) works with the transformations SubBytes, ShiftRows, MixColumns and AddRoundKey, all of them fixed and selected a priori. In this paper, we will show dynamic variants of AES, where the new transformations are RandomSubBytes, RandomShiftRows, RandomMixColumns and RandomAffineTransfKey.**

***Tóm tắt*—Thuật toán mã hóa AES (Tiêu chuẩn mã hóa nâng cao) bao gồm các phép biến đổi SubBytes, ShiftRows, MixColumn và AddRoundKey. Tất cả các phép biến đổi này đều cố định và được chọn ưu tiên. Trong bài báo này, nhóm tác giả sẽ trình bày một số biến thể động của AES, trong đó các phép biến đổi mới là RandomSubBytes, RandomShiftRows, RandomMixColumns và RandomAffineTransfKey.**

Keywords—Block cipher; AES; Dynamic transformations.

Từ khóa—Mã khối; AES; Phép biến đổi động.

I. INTRODUCTION

Rijndael is a cryptographic algorithm designed by the Belgian Joan Daemen and Vincent Rijmen and submitted to the AES competition in 1997. Announced as a winner in 2001, Rijndael was adopted as a standard [1] and named AES (Advanced Encryption Standard), with some specifications in terms of block and key sizes.

AES works with the fixed transformations SubBytes, ShiftRows, MixColumns and AddRoundKey, all of them selected a priori [2], [3]; however, a large number of AES variants with dynamic transformations depending on secret key can be seen in the specialized literature [4]–[18].

This manuscript is received on May 22, 2020. It is commented on May 28, 2020 and is accepted on August 14, 2020 by the first reviewer. It is commented on June 31, 2020 and is accepted on August 18, 2020 by the second reviewer.

Other cryptographic algorithms have been proposed with dynamic transformations in terms of greater security in their design. Two examples are: the block cipher Twofish [19], resulting a finalist in the AES competition, and the block cipher Grand Cru [17], submitted to the NESSIE process.

The aim of this paper is to show dynamic variants of the block cipher AES, working in all cases with the transformations *RandomSubBytes*, *RandomShiftRows*, *RandomMixColumns* and *RandomAffineTransfKey*, which are obtained at random from the set of all its possible choices.

We also present the algorithms for the random generation of the dynamic transformations cited above from pseudorandom sequences generated through the key schedule of AES or any other pseudorandom number generator, and we explain how the transformations RandomSubBytes and RandomAffinTransfKey represent two different approaches.

This work begins with a brief description of the cryptographic algorithm AES, continue with the explanation of the dynamic variants that we propose, as well as the dynamic transformations used in the rounds, and conclude with the presentation of the necessary algorithms for the random generation of these transformations.

Our contributions: In this paper, fully dynamic variants of the block cipher AES are presented, where all original transformations are replaced by key-dependent transformations, selected at random from the set of all its possible choices. A new algorithm for the random generation of MDS matrices in $GL_{4x4}(GF(2^8))$, and a random key-dependent affine transformation as an alternative variant for the key addition are presented.

II. THE BLOCK CIPHER AES

The operations in the cryptographic algorithm AES are performed in the Galois field $GF(2^8)$, so the input block and the output block are arrays of 16 bytes each one.

The bytes of the input block are located inside a matrix with 4 rows and 4 columns, named state matrix $S = (s_{i,j})$, so that for every input block $p_0 p_1 p_2, \dots, p_{15}$ we have $s_{i,j} = p_{i+4j}$, $0 \leq i,j < 4$. The last state is transformed into the output block $c_0 c_1 c_2 c_3, \dots, c_{15}$ in the inverse sense $c_{i+4j} = s_{i,j}$ for every $0 \leq i,j < 4$.

The secret key is another array of bytes, of size 16, 24 or 32, which is transformed into a matrix of 4 rows and N_k columns, where $N_k = 4$, $N_k = 6$ or $N_k = 8$ depending on the size of the key. The number of rounds N_r also depends on the size of the key, and it is computed as $N_r = N_k + 6$, where N_k rounds are added as security margin according to the criteria of the designers [3].

In each round a 16-byte-key is available. The 16-byte-keys are generated from the key schedule independently of the encryption process, and the following transformations act on the state matrix offering confusion and diffusion:

1. SubBytes, acting like an S-box on every byte of the state.
2. ShiftRows, performing cyclic rotations on the rows of the state.
3. MixColumns, multiplying every column of the state by one MDS matrix.
4. AddRoundKey, adding all bytes of the state with the round key.

The key schedule of AES can be seen in [2], [3] as well as any other detail of interest in its design criteria. We present next the pseudocode of the encryption process.

Encryption process of AES

```
AES(State, CipherKey)
{
    KeyExpansion(CipherKey, ExpandedKey);
    AddRoundKey(State, ExpandedKey[0]);
    for (i = 1; i < Nr; i++)
    {
        SubBytes(State);
        ShiftRows(State);
        MixColumns(State);
        AddRoundKey(State, ExpandedKey[i]);
    }
    SubBytes(State);
    ShiftRows(State);
    AddRoundKey(State, ExpandedKey[Nr]);
}
```

In AES, the S-box has been selected in such a way that the maximum correlation over it is at most 2^{-3} and the difference propagation probability is at most 2^{-6}. It is proven that the number of active S-boxes in four rounds of AES is lower bounded by 25, it gives us a minimum weight of 150 for any four-round differential trail, and a maximum correlation contribution of 2^{-75} for any four-round linear trail.

Hence, there are no eight-round trails with a weight below 300 or a correlation contribution less than 2^{-150}. The designers of AES consider this sufficient to resist differential and linear attacks; however, they added N_k extra rounds as security margin.

III. THE DYNAMIC AES

Dynamic encryption is a way to design block cipher algorithms, and AES has several dynamic variants in specialized literature. This idea is formally presented in [18] and constitutes a practical strength for block ciphers if the dynamic transformations satisfy the design requirements of the original transformations, since the dynamic algorithm has at least the same security as the original algorithm [18], [20].

In this section, we propose dynamic variants of the cryptographic algorithm AES, using the random transformations RandomSubBytes, RandomShiftRows, RandomMixColumns and RandomAffinTransfKey, as is shown in the next pseudocode.

Encryption process of the dynamic variants

```
AESDynamicVariant(State, CipherKey);
{
  KeyExpansion(CipherKey, ExpandedKey)
  RandomSubBytes(sequence_1, SubBytes)
  RandomShiftRows(sequence_2, ShiftRows)
  RandomMixColumns(sequence_3, MixColumns)
  RandomAffineTransfKey(sequence_4, TransfKey)
  AffineTransfKey(State, ExpandedKey[0])
  for (i = 1; i < N_r; i++)
    {
    SubBytes(State)
    ShiftRows(State)
    MixColumns(State)
    AffineTransfKey(State, ExpandedKey[i])
    }
  SubBytes(State)
  ShiftRows (State)
  AffineTransfKey(State, ExpandedKey[N_r])
}
```

Here sequence_i is a pseudorandom sequence for all $1 \leq i \leq 4$ that can be obtained from any pseudorandom number generator or directly from the AES key schedule.

A. RandomSubBytes

The random transformation RandomSubBytes is presented for the replacement of the fixed transformation SubBytes of AES, but it is derived into the two random transformations RandomMatrix and RandomPermutation with a different approach, each one of them used separately to provide confusion into the encryption process.

SubBytes acts like a S-box denoted as S_{DR} on every byte of the state, constructed through a non-affine transformation and an affine transformation. This S-box was selected by the AES designers taking in mind a complex algebraic expression; however, in the proposed dynamic variants, we consider to use a random S-box so that its algebraic expression will be unknown.

The first change that we propose is the generation of a random invertible matrix in $GL_{8x8}(GF(2))$ used to construct a key-dependent affine transformation, this way RandomMatrix composed with S_{DR} acts like a random S-box $RS_{DR}[x]$.

The second change is to construct a random S-box independent from S_{DR} through a random permutation Π of the symmetric group S_{256}, this way the transformation RandomPermutation acts like the random S-box $RS_{DR}[x] = \Pi[x]$.

Both, RandomMatrix and RandomPermutation, can be used without worrying on the cryptographic properties of the random S-box RS_{DR} if we use the encryption process in short plain texts or we use a rekeyed mode of operation for big plain texts. For the use of a block cipher in one of these modes, see for example [21] and [22].

B. RandomShiftRows and RandomMixColumns

The random transformations RandomShiftRows and RandomMixColumns are presented for the replacement of the fixed transformations ShiftRows and MixColumns of AES respectively, providing both diffusion as well as the original transformations.

ShiftRows acts on the rows of the state cyclically rotating their bytes 0, 1, 2 or 3 positions to the left respectively, providing dispersion between the columns of the state. The random ShiftRows is a random diffusion optimal permutation R of the symmetric group S_{16} [3] so that the bytes inside every column of the state are located into different columns after R.

MixColumns acts on the state multiplying every column by the MDS matrix

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

providing maximal local diffusion into the columns of the state. The random MixColumn is a random MDS matrix in $GL_{4x4}(GF(2^8))$.

C. RandomAffineTransfKey

In this paper, three ways to introduce the round key in the round function are proposed. The first way consists in a bitwise XOR like in AES, the second way consists in a random key-dependent affine transformation on the state of the form:

$$S \cdot M \oplus \text{ExpandedKey[t]}$$

where S is the state matrix and M is a random invertible matrix in $GL_{4x4}\big(GF(2^8)\big)$, and the third way consists in a random key-dependent affine transformation on every byte the state of the form:

$$L \cdot s_{i,j} \oplus \text{ExpandedKey[t]}_{i,j}$$

where L is a random invertible matrix in $GL_{8x8}(GF(2))$ for all $0 \leq i, j < 4$.

The first two ways of AffineTransfKey were also proposed in [23] for the block cipher SHARK and the pseudocode of the encryption process for the dynamic AES with these transformation was given above. In the third way of AffineTransfKey the two steps of the round function AffineTransfKey(State, ExpandedKey[t]) and SubBytes(State) can be computed as

$$(RS_{DR}\ o\ L) \cdot s_{i,j} \oplus L^{-1}(\ \text{ExpandedKey[t]}_{i,j})$$

for all $0 \leq i, j < 4$, then in terms of a more efficient encryption process for this case the pseudocode of the encryption process that we propose is:

Encryption process of the dynamic variant

AESDynamicVariant(State, CipherKey)

{

KeyExpansion(CipherKey, ExpandedKey)

RandomSubBytes(sequence_1, SubBytes)

RandomShiftRows(sequence_2, ShiftRows)

RandomMixColumns(sequence_3, MixColumns)

RandomMatrix(sequence_4, L)

AddRoundKey(State, L^{-1}(ExpandedKey[0]))

for (i = 1; i < N_r; i++)

{

(SubBytes o L) (State)

ShiftRows(State)

MixColumns(State)

AddRoundKey(State, L^{-1}(ExpandedKey[i]))

}

(SubBytes o L)(State)

ShiftRows (State)

AddRoundKey(State, L^{-1}(ExpandedKey[N_r]))

}

IV. The random transformations

The algorithms that we will present in this section allows the generation of RandomSubBytes, RandomShiftRows, RandomMixColumns and RandomAffineTransfKey, all of them are randomly generated from the set of all its possible choices.

The pseudorandom sequences used to compute these transformations are generated through the key schedule process or any other pseudorandom number generator. The inverse transformations are generated in a similar way.

A. Random Invertible Matrices

First, we present an algorithm for the random generation of invertible matrices in $GL_{8x8}\big(GF(2)\big)$, used for the transformations RandomMatrix, in place of SubBytes, and RandomAffineTransfKey.

The theoretical bases and complexity analysis for the random generation of such matrices and their inverses can be found in [24].

Input:

- Primitive polynomials $g_1(x), g_2(x), \dots, g_7(x)$ in $GF(2)[x]$ selected a priori so that $deg(g_1(x)) = 8$, $deg\big(g_2(x)\big) = 7, \dots,$ $deg(g_7(x)) = 2$.

- Pseudorandom binary sequence written as matrix

$$\begin{bmatrix} b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} & b_{1,6} & b_{1,7} \\ c_{2,0} & b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} & b_{2,6} \\ c_{3,0} & c_{3,1} & b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} \\ c_{4,0} & c_{4,1} & c_{4,2} & b_{4,0} & b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} \\ c_{5,0} & c_{5,1} & c_{5,2} & c_{5,3} & b_{5,0} & b_{5,1} & b_{5,2} & b_{5,3} \\ c_{6,0} & c_{6,1} & c_{6,2} & c_{6,3} & c_{6,4} & b_{6,0} & b_{6,1} & b_{6,2} \\ c_{7,0} & c_{7,1} & c_{7,2} & c_{7,3} & c_{7,4} & c_{7,5} & b_{7,0} & b_{7,1} \\ c_{8,0} & c_{8,1} & c_{8,2} & c_{8,3} & c_{8,4} & c_{8,5} & c_{8,6} & b_{8,0} \end{bmatrix}$$

where $b_{k,0}, b_{k,1}, \cdots, b_{k,8-k} \neq 0$ for all $1 \leq k \leq 8$.

{

Step 1: Computation of the first row

Input: $(a_0, a_1, a_2, \cdots, a_7) = (1, 0, \ldots, 0, 0)$

$$\hat{a}_0 + \hat{a}_1 x + \cdots + \hat{a}_7 x^7 = (a_0 + a_1 x + \cdots + a_7 x^7)\left(b_{1,0} + b_{1,1} x + \cdots + b_{1,7} x^7\right) mod\ g_1(x)$$

$(a_0, a_1, \cdots, a_7) = (\hat{a}_0, \hat{a}_1, \cdots, \hat{a}_7)$

Output: $row_1 = (a_0, a_1, \cdots, a_7)$

Step 2: Computation of the row $2 \leq j \leq 8$

Input: $(a_0, a_1, \cdots, a_7)$ the j – th canonical vector

for $(i = j; i > 1; i\text{--})$

{

$\hat{a}_0 = a_0 + c_{i,0} a_{i-1}$

$\hat{a}_1 = a_1 + c_{i,1} a_{i-1}$

$\vdots$

$\hat{a}_{i-2} = a_{i-2} + c_{i,i} a_{i-1}$

$$\hat{a}_{i-1} + \hat{a}_i x + \cdots + \hat{a}_7 x^{8-i} = (a_{i-1} + a_i x + \cdots + a_7 x^{8-i})\left(b_{i,0} + b_{i,1} x + \ldots + b_{i,8-i} x^{8-i}\right) mod\ g_i(x)$$

$(a_0, a_1, \cdots, a_7) = (\hat{a}_0, \hat{a}_1, \cdots, \hat{a}_7)$

}

Output: $row_j = (a_0, a_1, \ldots, a_7)$

}

Output:

Matrix $A = \begin{pmatrix} row_1 \\ row_2 \\ \vdots \\ row_8 \end{pmatrix}$

B. Random Permutations

Now we present an algorithm for the random generation of a permutation in the symmetric group S_{256} used for the transformation RandomPermutation in place of SubBytes. The theoretical bases and the complexity analysis for the random generation of a permutation in the symmetric group S_n can be found in [25].

Input:

Pseudorandom sequence $(\gamma_1, \gamma_2, \cdots, \gamma_{255})$ where $\gamma_i \in \{i, i+1, \cdots, 256\}$ for all $1 \leq i \leq 255$.

{

$\gamma_{256} = 256$

for $(j = 1; j < 256; j{+}{+})$

{

$\pi[j] = j$

for $(i = j; i > 0; i\text{--})$

{

$\pi[j] = (\pi[j] + \gamma_i - i)$

if $\pi[j] > 256$ then $\pi[j] = (\pi[j] + i - 1)\ mod\ 256$

}

}

}

Output:

Permutation $\Pi = \begin{pmatrix} 1 & 2 & \cdots & 255 \\ \pi[1] & \pi[2] & \cdots & \pi[255] \end{pmatrix}$

C. Random Diffusion Optimal Permutations

Next we present an algorithm for the random generation of a diffusion optimal permutation in the symmetric group S_{16}, used for the transformation RandomShiftRows in place of ShiftRows. The theoretical bases and the

complexity analysis for the random generation of such permutations can be found in [26].

Input:

Sequence of random permutations $R_0, R_1, \cdots, R_7$ where $R_i \in S_4$ for all $0 \leq i < 8$. (These can be generated with the algorithm described above.)

{

for $(i = 0; i < 4; i{+}{+})$

{

for $(j = 0; j < 4; j{+}{+})$

$R[4i + j] = 4(R_i[j]) + R_{4+R_i[j]}[i]$

}

}

Output:

Diffusion optimal permutation

$$R = \begin{pmatrix} 1 & 2 & \cdots & 16 \\ R[1] & R[2] & \cdots & R[16] \end{pmatrix}$$

Observations: The sequence of permutations R_i, where $0 \leq i < 8$, can be applied directly on the state in such a way that R_0, R_1, R_2 and R_3 are applied on the columns of the state, and once the resultant state is transposed, R_4, R_5, R_6 and R_7 are applied on the columns again.

D. Random MDS Matrices

In this subsection, we present two methods for the random generation of a MDS matrix in $GL_{4x4}\big(GF(2^8)\big)$. The complexity of both algorithms is very similar to the complexity of the algorithms presented in [24] for the generation of an invertible matrix and its inverse with elements in $GF(p^k)$, where p is prime and $k \in \mathbb{N}$.

We present first, the algorithms for the random generation of an invertible matrix in $GL_{4x4}(GF(2^8))$ and its inverse, which also will be used for RandomAffineTransfKey in place of AddRoundKey.

Computation of an invertible matrix:

Input:

- Primitive polynomials $g_1(x)$, $g_2(x)$, and $g_3(x)$ in $GF(2^8)[x]$ selected a priori so that $deg\big(g_1(x)\big) = 4$, $deg(g_2(x)) = 3$ and $deg(g_3(x)) = 2$.

- Pseudorandom sequence written as matrix

$$M = \begin{bmatrix} b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ c_{2,0} & b_{2,0} & b_{2,1} & b_{2,2} \\ c_{3,0} & c_{3,1} & b_{3,0} & b_{3,1} \\ c_{4,0} & c_{4,1} & c_{4,2} & b_{4,0} \end{bmatrix}$$

where $c_{i,j}, b_{k,t} \in GF(2^8)$ and $b_{k,0}, \cdots, b_{k,4-k} \neq 0$ for all $2 \leq i \leq 4$, $1 \leq k \leq 4$ and $0 \leq j \leq i - 2$, $0 \leq t \leq 4 - k$.

{

Step 1: Computation of the first row

Input: $(a_0, a_1, a_2, a_3) = (1, 0, 0, 0)$

$$\hat{a}_0 + \hat{a}_1 x + \hat{a}_2 x^2 + \hat{a}_3 x^3 = (a_0 + a_1 x + a_2 x^2 + a_3 x^3)\big(b_{1,0} + b_{1,1} x + b_{1,2} x^2 + b_{1,3} x^3\big) \bmod g_1(x)$$

$$(a_0, a_1, a_2, a_3) = (\hat{a}_0, \hat{a}_1, \hat{a}_2, \hat{a}_3)$$

Output: $row_1 = (a_0, a_1, a_2, a_3)$

Step 2: Computation of the row $2 \leq j \leq 4$

Input: (a_0, a_1, a_2, a_3) the j-th canonical vector

for $(i = j; i > 1; i{-}{-})$

{

$$\hat{a}_0 = a_0 + c_{i,0} a_{i-1}$$
$$\hat{a}_1 = a_1 + c_{i,1} a_{i-1}$$
$$\vdots$$
$$\hat{a}_{i-2} = a_{i-2} + c_{i,i-2} a_{i-1}$$

$$\hat{a}_{i-1} + \hat{a}_i x + \cdots + \hat{a}_3 x^{4-i} = (a_{i-1} + a_i x + \cdots + a_3 x^{4-i})\,(b_{i,0} + b_{i,1} x + \cdots + b_{i,4-i} x^{4-i}) \bmod g_i(x)$$

$$(a_0, a_1, a_2, a_3) = (\hat{a}_0, \hat{a}_1, \hat{a}_2, \hat{a}_3)$$

}

Output: $row_j = (a_0, a_1, a_2, a_3)$

}

Output:

Matrix $A = \begin{pmatrix} row_1 \\ row_2 \\ row_3 \\ row_4 \end{pmatrix}$

Computation of the inverse matrix:

Input:

- Primitive polynomials $g_1(x)$, $g_2(x)$ and $g_3(x)$ in $GF(2^8)[x]$ selected a priori so that $deg(g_1(x)) = 4$, $deg(g_2(x)) = 3$ and $deg(g_3(x)) = 2$.

- Pseudorandom sequence written as matrix

$$M = \begin{bmatrix} b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ c_{2,0} & b_{2,0} & b_{2,1} & b_{2,2} \\ c_{3,0} & c_{3,1} & b_{3,0} & b_{3,1} \\ c_{4,0} & c_{4,1} & c_{4,2} & b_{4,0} \end{bmatrix}$$

Where $c_{i,j}, b_{k,t} \in GF(2^8)$ and $b_{k,0}, \cdots, b_{k,4-k} \neq 0$ for all $2 \leq i \leq 4$, $1 \leq k \leq 4$ and $0 \leq j \leq i-2$, $0 \leq t \leq 4-k$.

{

Computation of the row $1 \leq j \leq 4$

Step 1:

Input: (a_0, a_1, a_2, a_3) the *j*-th canonical vector

$$\hat{a}_0 + \hat{a}_1 x + \hat{a}_2 x^2 + \hat{a}_3 x^3 = (a_0 + a_1 x + a_2 x^2 + a_3 x^3)\left(b_{1,0} + b_{1,1} x + b_{1,2} x^2 + b_{1,3} x^3\right)^{-1} mod\, g_1(x)$$

$(a_0, a_1, a_2, a_3) = (\hat{a}_0, \hat{a}_1, \hat{a}_2, \hat{a}_3)$

Output: (a_0, a_1, a_2, a_3)

Step 2:

Input: (a_0, a_1, a_2, a_3)

for $(i = 2; i \leq 4; i++)$

{

$$\hat{a}_{i-1} + \hat{a}_i x + \cdots + \hat{a}_3 x^{4-i} = (a_{i-1} + a_i x + \cdots + a_3 x^{4-i})\left(b_{i,0} + b_{i,1} x + \cdots + b_{i,4-i} x^{4-i}\right)^{-1} mod\, g_i(x)$$

$\hat{a}_0 = a_0 + c_{i,0} a_{i-1}$

$\hat{a}_1 = a_1 + c_{i,1} a_{i-1}$

$\vdots$

$\hat{a}_{i-2} = a_{i-2} + c_{i,i-2} a_{i-1}$

$(a_0, a_1, a_2, a_3) = (\hat{a}_0, \hat{a}_1, \hat{a}_2, \hat{a}_3)$

}

Output: $row_j = (a_0, a_1, a_2, a_3)$

}

Output:

Matrix $A^{-1} = \begin{pmatrix} row_1 \\ row_2 \\ row_3 \\ row_4 \end{pmatrix}$

Now we are able to present the algorithms necessary for the computation of a random MDS matrix for RandomMixColumns.

The first method:

Here we show as first method for the random generation of a MDS matrix a simpler presentation of the algorithm described in [27], where the next definition of MDS matrix has been used:

Any 4x4 matrix over $GF(2^n)$ *with all non-zero elements is a MDS matrix if and only if all its squares sub-matrices are not singular*.

Input:

- Primitive polynomials $g_1(x)$, $g_2(x)$ and $g_3(x)$ in $GF(2^8)[x]$ selected a priori so that $deg(g_1(x)) = 4$, $deg(g_2(x)) = 3$ and $deg(g_3(x)) = 2$.

- Pseudorandom sequence written as matrix

$$M = \begin{bmatrix} - & b_{1,1} & b_{1,2} & b_{1,3} \\ c_{2,0} & b_{2,0} & b_{2,1} & b_{2,2} \\ c_{3,0} & c_{3,1} & b_{3,0} & b_{3,1} \\ c_{4,0} & c_{4,1} & c_{4,2} & b_{4,0} \end{bmatrix}$$

where $c_{i,j}, b_{k,t} \in GF(2^8)$ and $b_{k,0}, \cdots, b_{k,4-k} \neq 0$ for all $2 \leq i \leq 4$, $2 \leq k \leq 4$ and $0 \leq j \leq 2$, $0 \leq t \leq 3$, and also $b_{1,1}$, $b_{1,2}$ and $b_{1,3} \neq 0$.

{

Step 1: Computation of the first row

The first row of a matrix A is formed by $b_{1,0}$, $b_{1,1}$, $b_{1,2}$ and $b_{1,3}$.Values $b_{1,1}$, $b_{1,2}$ and $b_{1,3}$ are taken from matrix M. The value $b_{1,0}$ will be determined in step 3 of the present algorithm.

Step 2: Computation of the row $2 \leq i \leq 4$

From the values of the first row, matrix M and the previous algorithm for random generation of an invertible matrix $A = \{a_{i,j}\}_{4x4}$, $a_{i,j} \in GF(2^8)$, the values $a_{i,j}$ are computed, leaving matrix A in the following way:

$$A = \begin{bmatrix} - & b_{1,1} & b_{1,2} & b_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,0} & a_{4,1} & a_{4,2} & a_{4,3} \end{bmatrix}$$

- The values $a_{i,j}$ are linear functions of $b_{1,0}$ and then if the values $a_{i,j}$ become equal to zero, linear equations with $b_{1,0}$ as unknown are formed.
- The determinants of all 2x2 sub-matrices are computed. If the determinants become equal to zero, then linear and quadratic equations with $b_{1,0}$ as unknown are formed.
- The determinants of all 3x3 sub-matrices are computed. If the determinants become equal to zero, then quadratic and cubic equations with $b_{1,0}$ as unknown are formed.
- The values of $b_{1,0}$ which do not satisfy the mentioned equations are stored.

Step 3: Random generation of a MDS matrix A

From the values of $b_{1,0}$ which do not satisfy the previous equations, one should be selected at random leaving matrix M full, and then matrix A is completed.

}

Output:

MDS matrix $A = \begin{bmatrix} b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,0} & a_{4,1} & a_{4,2} & a_{4,3} \end{bmatrix}$

Observations: If any of the values of the matrix A formed at the i-th row is zero, then a new value is randomly selected from matrix M, and the values $a_{i,j}$ for all $0 \leq j \leq 3$ are computed again.

The second method:

This new method proposed for the random generation of a MDS matrix uses the following proposition found in [28]:

Any 4x4 matrix over $GF(2^n)$ with all non-zero elements is a MDS matrix, if and only if it is full rank, the inverse matrix having all non-zero elements and all its 2x2 sub-matrices are full rank.

It is used the fact, if the polynomial $f(x) = b_{1,0} + b_{1,1}x + b_{1,2}x^2 + b_{1,3}x^3$ on $GF(2^8)$ is such that $b_{1,1}$, $b_{1,2}$ and $b_{1,3} \neq 0$ and $b_{1,0}$ is unknown, then all the coefficients of the inverse $f^{-1}(x)$ module $g(x)$, a primitive polynomial of degree 4, depend on $b_{1,0}$.

Input:

- Primitive polynomials $g_1(x)$, $g_2(x)$ and $g_3(x)$ in $GF(2^8)[x]$ selected a priori so that $deg(g_1(x)) = 4$, $deg(g_2(x)) = 3$ and $deg(g_3(x)) = 2$.

- Pseudorandom sequence written as matrix

$$M = \begin{bmatrix} - & b_{1,1} & b_{1,2} & b_{1,3} \\ c_{2,0} & b_{2,0} & b_{2,1} & b_{2,2} \\ c_{3,0} & c_{3,1} & b_{3,0} & b_{3,1} \\ c_{4,0} & c_{4,1} & c_{4,2} & b_{4,0} \end{bmatrix}$$

where $c_{i,j}, b_{k,t} \in GF(2^8)$ and $b_{k,0}, \cdots, b_{k,4-k} \neq 0$ for all $2 \leq i \leq 4$, $2 \leq k \leq 4$ and $0 \leq j \leq 2$, $0 \leq t \leq 3$, and also $b_{1,1}$, $b_{1,2}$ and $b_{1,3} \neq 0$.

{

Step 1: Computation of the first row

The first row of a matrix A is formed by $b_{1,0}$, $b_{1,1}$, $b_{1,2}$ and $b_{1,3}$. Values $b_{1,1}$, $b_{1,2}$ and $b_{1,3}$ are taken from matrix M. The value $b_{1,0}$ will be determined in step 6 of the present algorithm.

Step 2: Computation of the row $2 \leq i \leq 4$

From the values of the first row, matrix M and the previous algorithm for random generation of an invertible matrix $A = \{a_{i,j}\}_{4x4}$, $a_{i,j} \in GF(2^8)$, the values $a_{i,j}$ are computed, leaving matrix A in the following way:

$$A = \begin{bmatrix} - & b_{1,1} & b_{1,2} & b_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,0} & a_{4,1} & a_{4,2} & a_{4,3} \end{bmatrix}$$

- The values $a_{i,j}$ are linear functions of $b_{1,0}$ and then if the values $a_{i,j}$ become equal to zero linear equations with $b_{1,0}$ as unknown are formed.
- The determinants of all 2x2 sub-matrices are computed. If the determinants become equal to zero linear and quadratic equations with $b_{1,0}$ as unknown are formed.
- The values of $b_{1,0}$ which do not satisfy the mentioned equations are stored.

Step 3: With the coefficients $b_{1,1}$, $b_{1,2}$ and $b_{1,3}$ of matrix M and $b_{1,0}$ as unknown, it is computed the coefficients $d_{1,0} = \lambda_0(b_{1,0})$, $d_{1,1} = \lambda_1(b_{1,0})$, $d_{1,2} = \lambda_2(b_{1,0})$ and $d_{1,3} = \lambda_3(b_{1,0})$ of the inverse $f^{-1}(x) = d_{1,0} + d_{1,1}x + d_{1,2}x^2 + d_{1,3}x^3$ module $g_1(x)$. Thus, we can form the matrix

$$M' = \begin{bmatrix} d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ c_{2,0} & b_{2,0} & b_{2,1} & b_{2,2} \\ c_{3,0} & c_{3,1} & b_{3,0} & b_{3,1} \\ c_{4,0} & c_{4,1} & c_{4,2} & b_{4,0} \end{bmatrix}$$

Step 4: With matrix M' and the algorithm described above to compute the inverse we can compute

$$A^{-1} = \begin{bmatrix} d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\ d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\ d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3} \\ d_{4,0} & d_{4,1} & d_{4,2} & d_{4,3} \end{bmatrix}$$

Note that $d_{i,j}$ depends on $b_{1,0}$, $b_{1,1}$, $b_{1,2}$ and $b_{1,3}$ for all $2 \le i \le 4$ and $0 \le j \le 3$, then matrix A^{-1} turns

$$\begin{bmatrix} \delta_{1,0}(b_{1,0}) & \delta_{1,1}(b_{1,0}) & \delta_{1,2}(b_{1,0}) & \delta_{1,3}(b_{1,0}) \\ \delta_{2,0}(b_{1,0}) & \delta_{2,1}(b_{1,0}) & \delta_{2,2}(b_{1,0}) & \delta_{2,3}(b_{1,0}) \\ \delta_{3,0}(b_{1,0}) & \delta_{3,1}(b_{1,0}) & \delta_{3,2}(b_{1,0}) & \delta_{3,3}(b_{1,0}) \\ \delta_{4,0}(b_{1,0}) & \delta_{4,1}(b_{1,0}) & \delta_{4,2}(b_{1,0}) & \delta_{4,3}(b_{1,0}) \end{bmatrix}$$

Step 5: If the values of $\delta_{i,j}(b_{1,0})$ for all $1 \le i \le 4$ and $0 \le j \le 3$ become equal to zero, equations with $b_{1,0}$ as unknown are formed. The values of $b_{1,0}$ which do not satisfy the mentioned equations are stored.

Step 6: Random generation of a MDS matrix A.

From the values of $b_{1,0}$ which do not satisfy the equations of steps 2 and 5, one should be selected at random leaving matrix M full, and then matrix A and its inverse are completed.

}

Output:

MDS matrix $A = \begin{bmatrix} b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,0} & a_{4,1} & a_{4,2} & a_{4,3} \end{bmatrix}$

V. About the implementation

With the dynamic variants presented in this paper, the AES-NI instruction set would seem no longer be applicable; however, the different steps of the round function in these dynamic variants can be combined in look-up tables as recommended in [3], allowing for very fast implementation on processors with word lengths 32 or greater.

In AES, if we denote the input by a and the output by d, then for every column $0 \le j < 4$ of the state matrix we have

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = T_0\left[a_{0,j}\right] + T_1\left[a_{1,(j+1)\ mod\ 4}\right] + T_2\left[a_{2,(j+2)\ mod\ 4}\right] + T_3\left[a_{3,(j+3)\ mod\ 4}\right]$$

where the look-up tables T_0, T_1, T_2 and T_3 have each 256 four-bytes word and requiring 4KB of storage space. In this case the tables are performing as follow:

$$T_0[x] = \begin{bmatrix} 02S_{DR}[x] \\ 01S_{DR}[x] \\ 01S_{DR}[x] \\ 03S_{DR}[x] \end{bmatrix}; \quad T_1[x] = \begin{bmatrix} 03S_{DR}[x] \\ 02S_{DR}[x] \\ 01S_{DR}[x] \\ 01S_{DR}[x] \end{bmatrix}$$

$$T_2[x] = \begin{bmatrix} 01S_{DR}[x] \\ 03S_{DR}[x] \\ 02S_{DR}[x] \\ 01S_{DR}[x] \end{bmatrix}; \quad T_3[x] = \begin{bmatrix} 01S_{DR}[x] \\ 01S_{DR}[x] \\ 03S_{DR}[x] \\ 02S_{DR}[x] \end{bmatrix}$$

Here S_{DR} is the S-box of AES, working as the transformation SubBytes, then if we compute the dynamic S-box RS_{DR} and the dynamic MDS matrix $B = (b_{i,j})_{4x4}$ we can construct the key-dependent look-up tables

$$T_0[x] = \begin{bmatrix} b_{0,0}RS_{DR}[x] \\ b_{1,0}RS_{DR}[x] \\ b_{2,0}RS_{DR}[x] \\ b_{3,0}RS_{DR}[x] \end{bmatrix}$$

$$T_1[x] = \begin{bmatrix} b_{0,1}RS_{DR}[x] \\ b_{1,1}RS_{DR}[x] \\ b_{2,1}RS_{DR}[x] \\ b_{3,1}RS_{DR}[x] \end{bmatrix}$$

$$T_2[x] = \begin{bmatrix} b_{0,2}RS_{DR}[x] \\ b_{1,2}RS_{DR}[x] \\ b_{2,2}RS_{DR}[x] \\ b_{3,2}RS_{DR}[x] \end{bmatrix}$$

$$T_3[x] = \begin{bmatrix} b_{0,3}RS_{DR}[x] \\ b_{1,3}RS_{DR}[x] \\ b_{2,3}RS_{DR}[x] \\ b_{3,3}RS_{DR}[x] \end{bmatrix}$$

generated in the key schedule. This way the output d can be computed as

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = T_0\left[a_{R_0[j],R_{4+j}[0]}\right] + T_1\left[a_{R_1[j],R_{4+j}[1]}\right] + T_2\left[a_{R_2[j],R_{4+j}[2]}\right] + T_3\left[a_{R_3[j],R_{4+j}[3]}\right].$$

VI. Conclusion

In this paper, we presented dynamic variants of the cryptographic algorithm AES, in which the transformations SubBytes, ShiftRows, MixColumns, and AddRoundKey, all of them are fixed and selected a priori, were replaced by random key-dependent transformations, RandomSubBytes, RandomShiftRows, RandomMixColumns and RandomAffineTransfKey respectively.

These transformations can be chosen from the set of all its possible choices, and can be generated from pseudorandom sequences obtained through the key schedule of AES or any pseudorandom number generator.

In the case of the transformation SubBytes, we show two possible replacements of the S-box, and for the transformation AddRoundKey, we show three possible variants. On the other hand, we show two methods for the random generation of MDS matrices to replace the original MDS matrix used in the transformation MixColumns. The second method is new in specialized literature.

With the proposed changes, the design strategy of the block cipher AES was carefully fulfilled, RandomShiftRows is a random diffusion optimal permutation as the transformation ShiftRows, RandomMixColumns is a random MDS matrix like the transformation MixColumns. The random matrices used in RandomAffineTransfKey are invertible matrices and for the second proposal of this transformation we do not consider necessary adding extra round functions like in AES.

The most polemic change may be the random transformation RandomSubBytes, which can turn occasionally into a random S-box with bad properties contrary to SubBytes; however, we believe this is compensated by the unknowingness of the S-box and we recommend using a rekeyed block cipher mode of operation to encrypt long plain texts.

VII. Appendix

Inverse of a polynomial of degree 3

We present next an example of the inverse of a polynomial in $GF(2^8)[x]$ with degree 3, so that it can be used in the random generation of a MDS matrix through the second method described before.

Let $f(x)$ be the polynomial

$$x^3 + x^2 + x + z$$

and let $g(x)$ be the primitive polinomial

$$x^4 + z^9x^3 + z^2x + z^{13}$$

in $GF(2^8)[x]$, where $z \in GF(2^8)$ is a primitive element module the irreducible polynomial used to construct the field $GF(2^8)$ in Rijndael

$$P(y) = y^8 + y^4 + y^3 + y + 1$$

Let $f^{-1}(x) = d_{1,0} + d_{1,1}x + d_{1,2}x^2 + d_{1,3}x^3$ be the inverse of $f(x)$ module $g(x)$, so that $d_{1,0} = \lambda_0(b_{1,0})$, $d_{1,1} = \lambda_1(b_{1,0})$, $d_{1,2} = \lambda_2(b_{1,0})$, $d_{1,3} = \lambda_3(b_{1,0})$. If we take $z = y + 1$ then

$$d_{1,3} \equiv z^{137} \bmod(P(y))$$

$$d_{1,2} \equiv z^{215} \bmod(P(y))$$

$$d_{1,1} \equiv z^{61} \bmod(P(y))$$

$$d_{1,0} \equiv z^{155} \bmod(P(y))$$

REFERENCES

[1] Federal Information Processing Standard. Announcing the Advanced Encryption Standard (AES). FIPS Publication 197, 2001.

[2] Daemen J. and Rijmen V. "The Rijndael block cipher. AES proposal". 1999.

[3] Daemen J. and RijmenV. "The design of Rijndael: AES - The Advanced Encryption Standard". Second Edition. Springer. 2020.

[4] Fahmy A., Shaarawy M., El-Hadad K., Salama G. and Hassanain K. "A Proposal For A Key-Dependent AES". Proceedings of the SETIT-2005.

[5] Krishnamurthy G. and Ramaswamy V. "Making AES Stronger: AES with Key Dependent S-Box". International Journal of Computer Science and Network Security, Vol. 8, No. 9, 2008.

[6] El Ghafar A., Rohiem A., Diaa A. and Mohammed F. "Generation of AES Key Dependent S-Boxes using RC4 Algorithm". Proceedings of the ASAT-13, 2009.

[7] Hosseinkhani R. and Seyyed H. "Using Cipher Key to Generate Dynamic S-Box in AES Cipher System". International Journal of Computer Science and Security, Vol. 6, Issue 1, 2012.

[8] Ismail I., Galal-Edeen G., Khattab S. and Moustafa M. "Performance Examination of AES Encryption Algorithm with Constant and Dynamic Rotation". International Journal of Reviews in Computing, 2012.

[9] Ahmed F. and Elkamchouchi D. "Strongest AES with S-Boxes bank and dynamic key MDS matrix (SDK-AES)". International Journal of Computer and Communication Enginee-ring, Vol. 2, No. 4, 2013.

[10] Arrag S., Hamdoun A., Tragha A. and Khamlich S. "Implementation of Stronger AES by using Dynamic S-box Dependent of Master Key". Journal of Theoretical and Applied Information Technology, Vol. 53, No. 2, 2013.

[11] Freyre P, Díaz N and Cuellar O. "Variations to the cryptographic algorithms AES and Twofish". IACR e-print archive, No. 1080, 2015.

[12] Nidhinraj P. and George J. "DNA-based Approach of AES with Key Dependent ShiftRows". International Journal of Control Theory and Applications, Vol. 9, No. 43, 2016.

[13] Sachdeva, S. Doctoral dissertation "Improving AES-128 Using Multiple Cipher Keys and Key Dependent S-Boxes". Thapar Institute of Engineering and Technology, 2018.

[14] Al-Dweik, A., et al. "A Novel Method to Generate Key-Dependent S-Boxes with Identical Algebraic Properties." arXiv preprint arXiv:1908.09168. 2019.

[15] Partheeban, P. and Kavitha, V. "Dynamic key dependent AES S-box generation with optimized quality analysis". Cluster Computing, Vol. 22, Springer, 2019.

[16] Singh, A., Agarwal, P. and Chand, M. "Image Encryption and Analysis using Dynamic AES". 5th International Conference on Optimization and Applications ICOA. pp. 1-6, IEEE, 2019.

[17] Borst J. The block cipher: Grand Cru. available in: http://cryptonessie.org. Accessed on 01/9/2020.

[18] Knudsen L. Dynamic Encryption. Journal of Cyber Security. Vol. 3, 357-370, 2015.

[19] Schneier B. et al. "Twofish: A 128-bit block cipher". NIST AES Proposal, 15(1).1998.

[20] Rijmen V. Comment on dynamic encryption. available in: https://www.dencrypt.dk/wp-content/uploads/2017/05/Dencrypt-Vincent-Rijmen-opinion-on-Dynamic-Encryption.pdf. Accessed on 01/9/2020.

[21] Abdalla M. and Bellare M. "Increasing the life time of a key: a comparative analysis of the security of re-keying techniques". International Conference on the Theory and Application of Cryptology and Information Security. Springer, Berlin, Heidelberg, 2000.

[22] Lavrikov I. and Shishkin V. How much data may be safely processed on one key in different modes?. Mathematical Aspects of Cryptography. Vol. 10, 2019.

[23] Rijmen V., Daemen J., Preneel B., Bosselaers A. and De Win E. The cipher SHARK. LNCS 1039, pp. 99–111. Springer, 1996.

[24] Freyre P, Díaz N and Morgado E. R. "Some algorithms related to matrices with entries in a finite field". Journal of Discrete Mathematical Sciences & Cryptography. Vol. 12, No. 5, pp. 509–519. 2009.

[25] Freyre P and Díaz N. "Generación aleatoria de permutaciones del grupo simétrico o del grupo alternado". Revista Investigación Operacional. Vol. 36, No. 2, 2015.

[26] Alfonso A. and Freyre P. Random DiffusionOptimal Permutations with a Look in Dynamic Rijndael. Revista Ciencias Matemáticas. Vol. 32, 2018.

[27] Freyre P, Díaz N, Díaz R and Pérez C. "Random generation of MDS matrices". Proceedings of CurrentTrends in Cryptology CTCrypt2014. 2014.

[28] Gupta K. C. and Ray I. G. "On constructions of MDS matrices from companion matrices for lightweight cryptography". In CD-ARES.2013 Workshop: MOCrySEn, pp. 29-43, Springer. 2013.

ABOUT THE AUTHORS

PhD. Pablo Freyre Arrozarena

Workplace: Institute of Cryptography. University of Havana.

Email: pfreyre@matcom.uh.cu

Education process: Graduated of Mathematics in 1988; received Doctor's degree in 1998.

Current research direction: symmetric cryptography, mathematical aspects of cryptography and information security.

PhD. Oristela Cuellar Justiz

Workplace: Center for the Study of Computational Mathematics. University of Informatics Sciences.

Email: oristelacj@uci.cu

Education process: Graduated of Mathematics and Physics in 1987; received Doctor's degree in 2016.

Current research direction: symmetric cryptography, mathematical aspects of cryptography and information security.

MSc. Nelson Díaz Pérez

Workplace: Institute of Cryptography. University of Havana.

Education process: Graduated of Mathematics in 1985; received Master's degree in 2006.

Current research direction: symmetric cryptography, mathematical aspects of cryptography and information security.

MSc. Adrián Alfonso Peñate

Workplace: Institute of Cryptography. University of Havana.

Education process: Graduated of Mathematics in 2014; received Master's degree in 2018.

Current research direction: symmetric cryptography, mathematical aspects of cryptography and information security.