Hacking of the AES with Boolean Functions

Michel Dubois and Éric Filiol
Operational Cryptology and Virology Laboratory, 38 rue des Docteurs Calmette et Gurin, 53000 Laval, France

Keywords: Block Cipher, Boolean Function, Cryptanalysis, AES.

Abstract: One of the major issues of cryptography is the cryptanalysis of cipher algorithms. Some mechanisms for breaking codes include differential cryptanalysis, advanced statistics and brute-force. Recent works also attempt to use algebraic tools to reduce the cryptanalysis of a block cipher algorithm to the resolution of a system of quadratic equations describing the ciphering structure. In our study, we will also use algebraic tools but in a new way: by using Boolean functions and their properties. A Boolean function is a function from $F_2^n \to F_2$ with $n > 1$. The arguments of Boolean functions are binary words of length n. Any Boolean function can be represented, uniquely, by its algebraic normal form which is an equation which only contains additions modulo 2—the `XOR` function—and multiplications modulo 2—the `AND` function. Our aim is to describe the AES algorithm as a set of Boolean functions then calculate their algebraic normal forms by using the Moebius transforms. After, we use a specific representation for these equations to facilitate their analysis and particularly to try a combinatorial analysis. Through this approach we obtain a new kind of equations system.

1 INTRODUCTION

The block cipher algorithms are a family of cipher algorithms which use symmetric key and work on fixed length blocks of data.

Since Novembre 26, 2001, the block cipher algorithm "Rijndael", became the successor of DES under the name of "Advanced Encryption Standard" (AES). Its designers, Joan Daemen and Vincent Rijmen used algebraic tools to give to their algorithm an unequaled level of assurance against the standard statistical techniques of cryptanalysis. The AES can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits (NIST, 2001).

One of the major issues of cryptography is the cryptanalysis of cipher algorithms. Cryptanalysis is the study of methods for obtaining the meaning of encrypted information, without access to the secret information that is normally required. Some mechanisms for breaking codes include differential cryptanalysis, advanced statistics and brute-force.

Recent works like (Murphy and Robshaw, 2002), attempt to use algebraic tools to reduce the cryptanalysis of a block cipher algorithm to the resolution of a system of quadratic equations describing the ciphering structure. As an example, Nicolas Courtois and Josef Pieprzyk have described the AES-128 algorithm as a system of 8000 quadratic equations with 1600 variables (Courtois and Pieprzyk, 2002). Unfortunately, these approaches are infeasible because of the difficulty of solving large systems of equations.

We will also use algebraic tools but in a new way by using Boolean functions and their properties. Our aim is to describe a block cipher algorithm as a set of Boolean functions then calculate their algebraic normal forms by using the Moebius transforms.

In our study, we will test our approach on the AES algorithm. Our goal is to describe it under the form of systems of Boolean functions and to calculate their algebraic normal forms by using the Moebius transforms. The system of equations obtained is more easily implementable and could open new ways to cryptanalysis of the AES. We have developed a proof of concept of our approach in python language. The resulting programs are open source and available on github at the following address: https://github.com/archoad/BooleanAES.

2 BOOLEAN FUNCTIONS

2.1 Definition

Let be the set $B = \{0,1\}$ and $\mathcal{B}_2 = \{B, \wedge, \vee, \neg\}$ a Boolean algebra, then $\mathcal{B}_2^n = (x_1, x_2, \cdots, x_n)$ such that $x_i \in \mathcal{B}_2$ and $1 \leqq i \leqq n$, is a subset of $\mathcal{B}_2$ containing all n-tuples of 0 and 1. The variable x_i is called Boolean

Dubois, M. and Filiol, E.
Hacking of the AES with Boolean Functions.
DOI: 10.5220/0006091305990609
In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)*, pages 599-609
ISBN: 978-989-758-209-7

variable if she only accepts values from B, that is to say, if and only if $x_i = 0$ or $x_i = 1$ regardless of $1 \leqq i \leqq n$.

A Boolean function of degree n with $n > 1$ is a function f defined from $\mathcal{B}_2^n \to \mathcal{B}_2$, that is to say built from Boolean variables and agreeing to return values only in the set $B = \{0,1\}$.

For example, the function $f(x_1,x_2) = x_1 \wedge \neg x_2$ defined from $\mathcal{B}_2^2 \to \mathcal{B}_2$ is a Boolean function of degree two with:

$$f(0,0) = 0$$
$$f(0,1) = 0$$
$$f(1,0) = 1$$
$$f(1,1) = 0$$

Let n and m be two positive integers. A vector Boolean function is a Boolean function f defined from $\mathcal{B}_2^n \to \mathcal{B}_2^m$.

Finally, we can define a random Boolean function as a Boolean function f whose values are independent and identically distributed random variables, that is to say:

$$\forall (x_1,x_2,\cdots,x_n) \in \mathcal{B}_2^n$$
$$P[f(x_1,x_2,\cdots,x_n) = 0] = \frac{1}{2}$$

The number of Boolean functions is limited and depends on n. Thus, there is 2^{2^n} Boolean functions. Similarly, the number of vector Boolean functions is limited and depends on n and m. Thus, there exists $(2^m)^{2^n}$ vector Boolean functions.

If we take, for example, $n = 2$ then there exists $\left(2^2\right)^2 = 16$ Boolean functions of degree two. These 16 Boolean functions are presented in the table 1. Among the Boolean functions of degree 2, the best known are the functions OR, AND and XOR.

The support $supp(f)$ of a Boolean function is the set of elements x such that $f(x) \neq 0$, the Hamming weight $wt(f)$ of a Boolean function is the cardinal from its support and we have:

$$wt(f) = |\{x \in \mathcal{B}_2^n \quad | \quad f(x) = 1\}|$$

A Boolean function is called balanced if $wt(f) = 2^{n-1}$. Similarly, a Boolean vector function $\mathcal{B}_2^n \to \mathcal{B}_2^m$ is said to be balanced if $wt(f) = 2^{n-m}$ (Carlet, 2010b).

For example, the support of the function $f(x_1,x_2) = x_1 \vee x_2$, corresponding to logical OR is $supp(f) = \{(0,1),(1,0),(1,1)\}$ and its weight is $wt(\mathrm{f}) = 3$.

Table 1: The 16 Boolean functions of degree 2.

f_0	0
f_1	$x_1 \wedge x_2$
f_2	$x_1 \wedge \neg x_2$
f_3	x_1
f_4	$\neg x_1 \wedge x_2$
f_5	x_2
f_6	$x_1 \veebar x_2$
f_7	$x_1 \vee x_2$
f_8	$\neg(x_1 \vee x_2)$
f_9	$\neg(x_1 \veebar x_2)$
f_{10}	$\neg x_2$
f_{11}	$x_1 \vee \neg x_2$
f_{12}	$\neg x_1$
f_{13}	$\neg x_1 \vee x_2$
f_{14}	$\neg(x_1 \wedge x_2)$
f_{15}	1

2.2 Representations

There are multiple representations of Boolean functions. We'll look at the most common—the truth table—and that we will use later—a representation in $GF(2)$.

2.2.1 The Truth Table

The different values taken by a Boolean function may be presented in the form of a table called truth table. The truth table characterizes a Boolean function.

2.2.2 Representation in GF(2)

A Boolean function can also be presented in the form of a series of conjunctions including disjunctions, negations and/or variables. This is called the conjunctive normal form. Thus, the sequence $f = (a \vee b) \wedge (\neg a \vee b)$ is the conjunctive normal form of the f function. Conversely, a Boolean function can be presented in the form of a series of disjunctions including conjunctions, negations and/or variables. This is called the disjunctive normal form. Thus, the sequence $g = (a \wedge b) \vee (\neg a \wedge b)$ is the disjunctive normal form of the function g.

Now let the representation of Boolean functions in $GF(2)$.

The set $B = \{0,1\}$ associated with $\wedge$, $\vee$ and $\neg$ operations is the Boolean algebra $\mathcal{B}_2 = \{B, \wedge, \vee, \neg\}$ with the truth tables of the operations described in figure (see fig. 1). If we introduce the two binary operations $\oplus$ and $\bullet$ defined by the truth tables in figure (see fig. 2), then $\mathcal{B}_2$ and the Galois field $GF(2)$ are similar. More specifically, the Boolean algebra

$\wedge$	0	1
0	0	0
1	0	1

$\vee$	0	1
0	0	1
1	1	1

a	0	1
$\neg a$	1	0

Figure 1: Rules for Boolean algebra with two elements.

$\bullet$	0	1
0	0	0
1	0	1

$\oplus$	0	1
0	0	1
1	1	0

Figure 2: Truth tables of $\bullet$ and $\oplus$.

$(B,\wedge,\vee,\neg)$ and the field $(GF(2),\bullet,\oplus)$ are related by the following transformation formulas:

$$\begin{aligned} a\wedge b &= a\bullet b \\ a\bullet b &= a\wedge b \\ a\vee b &= a\oplus b\oplus(a\bullet b) \\ a\oplus b &= (a\wedge\neg b)\vee(\neg a\wedge b) \\ \neg a &= a\oplus 1 \end{aligned}$$

We can now define a Boolean function as a function $f:\mathbb{F}_2^n\to\mathbb{F}_2$ with $\mathbb{F}_2^n$ the set of binary vectors of length $n>1$. The Hamming weight $wH(x)$ of the binary vector $x\in\mathbb{F}_2^n$ is the number of non-zero coordinates, that is to say the size of the set $\{i\in\mathbb{N} \mid x_i\neq 0\}$. The Hamming weight of a Boolean function $f:\mathbb{F}_2^n\to\mathbb{F}_2$ is the size of its support. Finally, the Hamming distance between two Boolean functions f and g is the size of the set $\{x\in\mathbb{F}_2^n \mid f(x)\neq g(x)\}$.

Among the classic representation of Boolean functions, the most frequently used in cryptography is the polynomial representation in n-variable on $GF(2)$. This representation is of the form (Carlet, 2010a):

$$\begin{aligned} f(x) &= \bigoplus_{I\in P(N)} a_I\left(\prod_{i\in I} x_i\right) \\ &= \bigoplus_{I\in P(N)} a_i x^I \end{aligned}$$

$P(N)$ denotes the set of powers of $N=\{1,\cdots,n\}$. Each coordinate x_i appears in this polynomial with an exponent equal to at least one, because in $\mathbb{F}_2$ we have$x^2=x$. This representation is described in $\mathbb{F}_2[x_1,\cdots,x_n]/(x_1^2\oplus x_1,\cdots,x_n^2\oplus x_n)$.

This representation of Boolean functions in $GF(2)$ is called Reed-Muller expansion or polynomials of Zhegalkin ((O'Donnel, 2014) page 169) or, more commonly, *algebraic normal form* (ANF). The degree of $ANF(f)$ is the highest degree of monomials of $ANF(f)$ with non-zero coefficients. Finally, the algebraic normal form of a Boolean function exists and is unique.

In summary, any Boolean function can be represented uniquely by its algebraic normal form as the equation:

$$\begin{aligned} f(x_1,\cdots,x_n) = a_0+ & \\ & a_1x_1+a_2x_2+\cdots+a_nx_n+ \\ & a_{1,2}x_1x_2+\cdots+a_{n-1,n}x_{n-1}x_n+ \\ & \cdots+ \\ & a_{1,2,\ldots,n}x_1x_2\ldots x_n \end{aligned}$$

Consider an example. Let the function f described by the truth table of the table 2

Table 2: Truth table of the function f.

x_1	x_2	x_3	$f(x)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

The weight of the function f is $wt(f)=3$. So we can reduce f to the sum of 3 atomic functions f_1, f_2 and f_3. The function $f_1=1$ if and only if $1\oplus x_1=1$, $1\oplus x_2=1$ and $x_3=1$. From this we can deduce that the ANF of the function f_1 can be obtained by expanding the product $(1\oplus x_1)(1\oplus x_2)x_3$. Applying this reasoning to the functions f_2 and f_3 we get the following equation:

$$\begin{aligned} ANF(f) &= (1\oplus x_1)(1\oplus x_2)x_3\oplus x_1(1\oplus x_2)x_3\oplus x_1x_2x_3 \\ &= x_1x_2x_3\oplus x_1x_3\oplus x_3 \end{aligned}$$

3 MECHANISM OF THE EQUATIONS

After this brief presentation of Boolean functions, we have the necessary tools for the development of systems of Boolean equations describing the *Advanced Encryption standard.*

3.1 Moebius Transform

We have just seen how to generate normal algebraic form (ANF) of a Boolean function. The presented

method is not easily automatable in a computer program. So we will prefer the use of the Moebius transform.

The Moebius transform of the Boolean function f is defined by (McCarty, 1986):

$$TM(f) : \mathbb{F}_2^n \to \mathbb{F}_2$$
$$u = \bigoplus_{v \leqslant u} f(v) \text{mod} 2$$

with $v \leqslant u$ if and only if $\forall i, v_i = 1 \Rightarrow u_i = 1$.

From there, we can define the normal algebraic form of a Boolean function f in n variables:

$$\bigoplus_{u=(u_1,\cdots,u_n)\in\mathbb{F}_2^n} TM(u) x_1^{u_1} \cdots x_n^{u_n}$$

To better understand the mechanisms involved in the use of the Moebius transform, take an example with the `MajParmi3`. This function from $\mathbb{F}_2^3 \to \mathbb{F}_2$ is characterized by the truth table 3.

Table 3: The truth table of the function `MajParmi3`.

x_1	x_2	x_3	MP3
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Calculating the Moebius transform of the function we get the result of the table 4.

Table 4: Calculating the Moebius transform for `MajParmi3`.

x_1	x_2	x_3	compute of $mt(f)$						$mt(f)$
0	0	0	0	$\to$	0	0	0	0	0
0	0	1	0	$\to$	0	0	0	0	0
0	1	0	0	$\to$	0	0	0	0	0
0	1	1	1	$\to$	1	1	1	1	1
1	0	0	0	$\to$	0	0	0	0	0
1	0	1	1	$\to$	1	1	1	1	1
1	1	0	1	$\to$	1	1	1	1	1
1	1	1	1	$\to$	1	0	1	0	0

After the Moebius transform of the function obtained, we take the $\mathbb{F}_2^3$ for which $TM(\text{MajParmi3}) \neq 0$. In our case we have the triplets $(0,1,1)$, $(1,0,1)$, $(1,1,0)$ from which we can deduce the equation:

$$\text{MajParmi3}(x_1,x_2,x_3) = x_2x_3 + x_1x_3 + x_1x_2$$

With the addition corresponding to a `XOR` and multiplication to a `AND`.

$$f(b_1) = 1 \oplus b_{15}b_{16} \oplus b_{14} \oplus b_{14}b_{16} \oplus b_{13} \oplus b_{13}b_{15} \oplus b_{13}b_{15}b_{16} \oplus b_4 \oplus b_3b_4 \oplus b_2b_4 \oplus b_2b_3 \oplus b_2b_3b_4 \oplus b_1b_3 \oplus b_1b_3b_4 \oplus b_1b_2 \oplus b_1b_2b_3$$

1	1	0000000000000000
$b_{15}b_{16}$	0	0000000000000011
b_{14}	0	0000000000000100
$b_{14}b_{16}$	0	0000000000000101
b_{13}	0	0000000000001000
$b_{13}b_{15}$	0	0000000000001010
$b_{13}b_{15}b_{16}$	0	0000000000001011
b_4	0	0001000000000000
b_3b_4	0	0011000000000000
b_2b_4	0	0101000000000000
b_2b_3	0	0110000000000000
$b_2b_3b_4$	0	0111000000000000
b_1b_3	0	1010000000000000
$b_1b_3b_4$	0	1011000000000000
b_1b_2	0	1100000000000000
$b_1b_2b_3$	0	1110000000000000

Figure 3: File for the bit b_1.

3.2 Formatting Equations

To facilitate the analysis and in particular to try a combinatorial study we will implement a specific presentation for equations thus obtained.

The AES algorithm takes 128 bits as input and provides 128 bits as output. So we will have Boolean functions $F_2^{128} \to F_2^{128}$. The guiding principle is to generate a file by bit, we will have at the end 128 files. Each file containing the Boolean equation of the concerned bit.

In each file, the Boolean equation is presented under the form of lines containing sequences of 0 and 1. Each line describes a monomial of the equation and the transition from one line to another means applying a `XOR`.

In order to facilitate understanding of the chosen mechanism we describe the realization of file corresponding to one bit b_1 from his equation to the file formalism in figure (see fig. 3).

4 APPLICATION TO AES

4.1 Presentation of the AES

Since November 26, 2001, the block encryption algorithm "Rijndael", in its 128-bit version, became the DES successor under the name of *Advanced Encryption Standard* (AES).

Issued from a competition launched by the National Institute of Standards and Technology (NIST)

in 1997, Rijndael (Daemen and Rijmen, 1999) has crossed all the stages of selection and is now a U.S. federal standard recorded under the FIPS 197 number (NIST, 2001). Inscribed on the suite B of the National Security Agency (NSA), the AES is intended, promoted by the U.S. Government, to become a standard for secure exchange of classified information, into the United States and between the United States and their partners (CNSS, 2012). Indeed, originally reserved for the encryption of sensitive unclassified information—article 6 of (NIST, 2001)—the scope of the AES has evolved and became effective October first, 2015, the encryption algorithm for the information classified up to TOP secret in the United States—Annex B of (CNSS, 2012). Similarly, it is, today, the symmetric block cipher algorithm most commonly used in occident[1].

AES is a symmetric block cipher algorithm. It encrypts and decrypts data from one key blocks.

Unlike the DES, based on a Feistel network, the AES relies on a network of substitutions and permutations (SP-network). The latter consists of non-linear substitution functions contained in one S-Box and linear permutation functions we can be group into a P-Box. Each box takes a block of text and the key as input and return a block of ciphertext as output. The information flow in a set of several P-Box and S-Box suite forming a round.

The inputs and outputs of the AES are 128-bit blocks and the length of the key can be 128, 192 or 256 bits. The basic unit of the algorithm is the byte. Input data blocks are converted into tables of four columns and four rows, each box containing a byte, i.e. $4*4*8 = 128$ bits per table.

For encryption and decryption operations, the AES algorithm uses a function of round composed of four different functions. The first performs a substitution of bytes using a substitution table or S-box, the second executes a sliding of the rows of the states array from different offsets, the third performs a mixture of the columns of the states array and finally, the fourth adds the round key to the states array. The second and the third function form the P-box of the round.

The ciphering operations rely on four predefined functions: `AddRoundKey`, `SubBytes`, `ShiftRows` and `MixColumns`. Each of these functions is performed on the states array. Encryption cycle includes an initial transformation, some intermediate rounds and a final round.

The number of rounds in the AES is dependent on the key size. Thus, for a 128-bit key, the number of rounds is 10. Similarly, we have 12 rounds for a 192-bit key and 14 rounds for a 256-bit key.

In the end, the pseudo code of the AES encryption function can be written as follows, `Nb` corresponding to the 32-bits words number and `Nr` corresponding to the rounds number used in the algorithm.

```
1: function CIPHER(byte in[4*Nb], byte out[4*Nb], word
   w[Nb*(Nr+1)])
2:     byte state[4,Nb]
3:     state ← in
4:     AddRounkey(state, w[0, Nb-1])
5:     for round=1 step 1 to Nr-1 do
6:         SubBytes(state)
7:         ShiftRows(state)
8:         MixColumns(state)
9:         AddRoundKey(state, w[round*Nb,
   (round+1)*Nb-1])
10:    end for
11:    SubBytes(state)
12:    ShiftRows(state)
13:    AddRounkey(state, w[Nr*Nb, (Nr+1)*Nb-1])
14:    return state
15: end function
```

4.2 The Equations for AES

We will now apply to the AES the mechanism described above. The difficulty with our approach is that the encryption functions of the AES algorithm takes 128 bits as input and provides 128 bits as output. So we will have Boolean functions $F_2^{128} \to F_2^{128}$ and it is impossible to calculate their truth tables. Indeed, in this case, we have $2^{128} = 3,402823 \times 10^{38}$ possible combinations of 128-bit blocks and the space storage needed to archive these blocks is $3,868562 \times 10^{25}$ terabytes.

So we have to find a way to describe the AES encryption functions in the form of Boolean functions without using their truth table.

4.3 The Equations for Ciphering Functions

We will now detail the solution implemented for each of the sub-functions of the AES encryption algorithm.

4.3.1 Solution for SubBytes Function

The function `SubBytes` is a non-linear substitution that works on every byte of the states array using a substitution table (S-Box).

This function is applied independently to each byte of the input block. So, the S-box of the AES is a function taking 8 bits as input and providing 8-bit

[1]Russia, for example, uses encryption algorithms defined by the standards GOST 28147-89, GOST R 34.10 - 2001, etc...

as output. So we can describe it as a Boolean function $F_2^8 \rightarrow F_2^8$. From there, we can calculate the truth table of the S-Box and use the Moebius transform for obtain the normal algebraic form of the S-Box. Then applying the results to the 16 bytes of input block, we get 128 equations, each describing a block bit.

4.3.2 Solution for ShiftRows Function

In the `ShiftRows` function, the bytes of the third column of the state table are shifted cyclically in an offset whose size is dependent on the line number. The bytes of the first line do not suffer this offset.

For this function, we do not need to calculate specific Boolean function. Indeed, the only change made consists to shift bytes in the states array. In our files, this transformation can be easily solved by using a `XOR`.

Thus, for example, the second byte of the status table becomes the sixth byte after the application of `ShiftRows`. In the end, the equations of the function `ShiftRows` for the 128-bit of the block $B = (b_0 \dots b_{127})$ are:

$$(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_{40}, x_{41}, x_{42}, x_{43}, x_{44}, x_{45}, x_{46},$$
$$x_{47}, x_{80}, x_{81}, x_{82}, x_{83}, x_{84}, x_{85}, x_{86}, x_{87}, x_{120}, x_{121}, x_{122}, x_{123},$$
$$x_{124}, x_{125}, x_{126}, x_{127}, x_{32}, x_{33}, x_{34}, x_{35}, x_{36}, x_{37}, x_{38}, x_{39}, x_{72},$$
$$x_{73}, x_{74}, x_{75}, x_{76}, x_{77}, x_{78}, x_{79}, x_{112}, x_{113}, x_{114}, x_{115}, x_{116}, x_{117},$$
$$x_{118}, x_{119}, x_{24}, x_{25}, x_{26}, x_{27}, x_{28}, x_{29}, x_{30}, x_{31}, x_{64}, x_{65}, x_{66},$$
$$x_{67}, x_{68}, x_{69}, x_{70}, x_{71}, x_{104}, x_{105}, x_{106}, x_{107}, x_{108}, x_{109}, x_{110},$$
$$x_{111}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20}, x_{21}, x_{22}, x_{23}, x_{56}, x_{57}, x_{58}, x_{59}, x_{60},$$
$$x_{61}, x_{62}, x_{63}, x_{96}, x_{97}, x_{98}, x_{99}, x_{100}, x_{101}, x_{102}, x_{103}, x_8, x_9,$$
$$x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{48}, x_{49}, x_{50}, x_{51}, x_{52}, x_{53}, x_{54},$$
$$x_{55}, x_{88}, x_{89}, x_{90}, x_{91}, x_{92}, x_{93}, x_{94}, x_{95})$$

4.3.3 Solution for MixColumns Function

The function `MixColumns` acts on the states array, column by column, treating each column as a polynomial with four terms. Each column is multiplied by a square matrix. For each column we have:

$$\begin{pmatrix} b'_i \\ b'_{i+1} \\ b'_{i+2} \\ b'_{i+3} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \bullet \begin{pmatrix} b_i \\ b_{i+1} \\ b_{i+2} \\ b_{i+3} \end{pmatrix}$$

Thus, for the first byte of the column we have the equation:

$$b'_i = 02 \bullet b_i \oplus 03 \bullet b_{i+1} \oplus 01 \bullet b_{i+2} \oplus 01 \bullet b_{i+3}$$

As in GF_2^8, 01 is the identity for multiplication, this equation becomes:

$$b'_i = 02 \bullet b_i \oplus 03 \bullet b_{i+1} \oplus b_{i+2} \oplus b_{i+3}$$

We have the same simplification for all equations describing the multiplication of the column of the states array by the square matrix. Therefore we only need to calculate truth tables for multiplication by 02 and 03 in GF_2^8.

For example, the equations of the bits b_{120} to b_{127} are the following:

$$b_{120} = x_{97} \oplus x_{96} \oplus x_{104} \oplus x_{112} \oplus x_{121}$$
$$b_{121} = x_{98} \oplus x_{97} \oplus x_{105} \oplus x_{113} \oplus x_{122}$$
$$b_{122} = x_{99} \oplus x_{98} \oplus x_{106} \oplus x_{114} \oplus x_{123}$$
$$b_{123} = x_{100} \oplus x_{99} \oplus x_{96} \oplus x_{107} \oplus x_{115} \oplus x_{124} \oplus x_{120}$$
$$b_{124} = x_{101} \oplus x_{100} \oplus x_{96} \oplus x_{108} \oplus x_{116} \oplus x_{125} \oplus x_{120}$$
$$b_{125} = x_{102} \oplus x_{101} \oplus x_{109} \oplus x_{117} \oplus x_{126}$$
$$b_{126} = x_{103} \oplus x_{102} \oplus x_{96} \oplus x_{110} \oplus x_{118} \oplus x_{127} \oplus x_{120}$$
$$b_{127} = x_{103} \oplus x_{96} \oplus x_{111} \oplus x_{119} \oplus x_{120}$$

4.3.4 Solution for The Key Expansion Function

To recall, in the algorithm of the AES-128, Nb $= 4$ words and Nr $= 10$ words, with 1 word = 4 bytes = 32 bits.

The function `AddRoundKey` adds a round key to the state table by a simple bitwise `XOR` operation. These rounds keys are computed by a key expansion function. This latter generates a set of $Nb(Nr+1) =$ 44 words of 32 bit that to say 11 keys of 128 bits derived from the first key. The algorithm used for the expansion of the key involves two functions `SubWord` and `RotWord` together with a round constant `Rcon`.

The generation of a global Boolean function for the key expansion algorithm is impossible because the generation of the key for the round n involves the key of the round $n-1$. This interweaving of rounds keys does not allow us to generate a global Boolean function. On the other hand it is possible to generate a Boolean function corresponding to the calculation of a key of one round.

The first word w_{i_0} of the round key i is calculated according to the following equation:

$$w_{i_0} = (SW \circ RW(w_{(i-1)_3})) \oplus Rcon_i \oplus w_{(i-1)_0}$$

with $SW()$ and $RW()$ respectively corresponding to the `SubWord` and `RotWord` functions.

The following words w_{i_1}, w_{i_2} and w_{i_3} are calculated according to the following equation:

$$w_{i_n} = w_{i_{n-1}} \oplus w_{(i-1)_n}$$

with $1 \leq n \leq 3$.

The `SubWord` and `RotWord` functions are built on the same principle as the `SubBytes` and `ShiftRows` functions, thus we can reuse the methodology finalized previously.

In python language, the word generation function is written according to the following code:

```
def generateWord(num):
  if (num < 4):
    w = generateGenericWord(wordSize*num, 'x')
  if (num >= 4):
    if ((num % 4) == 0):
      w = generateWord(3)
      w = rotWord(w)
      w = subWord(w, rconList[(num/4)-1])
      w = xorWords(w, generateWord(0))
    else:
      w = generateWord(num-1)
      w = xorWords(w, generateWord(num%4))
  return w
```

In this code, several scenarios are considered. The function `generateWord` takes in parameter the word number to generate, we know that this number is between 0 and 43. If the number is less than 4, the function returns the Boolean identity function as the first key used by the AES is the encryption key. If the number to modulo 4 is zero, the function returns a Boolean functions describing the composition of `SubWord` and `RotWord` functions and the application of the `XOR` with the `Rcon` constant. Finally, if the number to modulo 4 is not zero, the function returns the Boolean function describing the `XOR` with the corresponding word in the previous round.

We now have a Boolean function describing a round expansion of the key. As we have seen, the key expansion algorithm involves at round n the keys of round $n-1$. To integrate our Boolean function in the encryption process of the AES, we must, at every round, add a temporary variable corresponding to the key of the previous round.

As an example, the Boolean equation of the bit b_0 of the fourth word on the 44 words generate by the key expansion process, is given below:

$$
\begin{aligned}
&x_{109} \oplus x_{109}x_{111} \oplus x_{109}x_{110} \oplus x_{108}x_{109}x_{111} \oplus x_{108}x_{109}x_{110} \oplus \\
&x_{108}x_{109}x_{110}x_{111} \oplus x_{107} \oplus x_{107}x_{110}x_{111} \oplus x_{107}x_{109} \oplus \\
&x_{107}x_{109}x_{110}x_{111} \oplus x_{107}x_{108}x_{110}x_{111} \oplus x_{107}x_{108}x_{109}x_{110} \oplus \\
&x_{107}x_{108}x_{109}x_{110}x_{111} \oplus x_{106} \oplus x_{106}x_{110}x_{111} \oplus x_{106}x_{109}x_{111} \oplus \\
&x_{106}x_{109}x_{110}x_{111} \oplus x_{106}x_{108} \oplus x_{106}x_{108}x_{111} \oplus x_{106}x_{108}x_{110} \oplus \\
&x_{106}x_{108}x_{109} \oplus x_{106}x_{108}x_{109}x_{111} \oplus x_{106}x_{108}x_{109}x_{110} \oplus \\
&x_{106}x_{107}x_{111} \oplus x_{106}x_{107}x_{109}x_{110} \oplus x_{106}x_{107}x_{108} \oplus \\
&x_{106}x_{107}x_{108}x_{110}x_{111} \oplus x_{106}x_{107}x_{108}x_{109}x_{111} \oplus x_{105}x_{111} \oplus \\
&x_{105}x_{110}x_{111} \oplus x_{105}x_{109} \oplus x_{105}x_{109}x_{110} \oplus x_{105}x_{108}x_{111} \oplus \\
&x_{105}x_{108}x_{110} \oplus x_{105}x_{108}x_{110}x_{111} \oplus x_{105}x_{108}x_{109}x_{111} \oplus \\
&x_{105}x_{108}x_{109}x_{110}x_{111} \oplus x_{105}x_{107} \oplus x_{105}x_{107}x_{109} \oplus \\
&x_{105}x_{107}x_{109}x_{111} \oplus x_{105}x_{107}x_{109}x_{110} \oplus \\
&x_{105}x_{107}x_{109}x_{110}x_{111} \oplus x_{105}x_{107}x_{108}x_{111} \oplus \\
&x_{105}x_{107}x_{108}x_{109}x_{111} \oplus x_{105}x_{106}x_{111} \oplus x_{105}x_{106}x_{109} \oplus \\
&x_{105}x_{106}x_{108}x_{111} \oplus x_{105}x_{106}x_{108}x_{109}x_{110} \oplus x_{105}x_{106}x_{107} \oplus \\
&x_{105}x_{106}x_{107}x_{110}x_{111} \oplus x_{105}x_{106}x_{107}x_{109}x_{110} \oplus \\
&x_{105}x_{106}x_{107}x_{108} \oplus x_{105}x_{106}x_{107}x_{108}x_{111} \oplus \\
&x_{105}x_{106}x_{107}x_{108}x_{109} \oplus x_{105}x_{106}x_{107}x_{108}x_{109}x_{111} \oplus x_{104} \oplus \\
&x_{104}x_{111} \oplus x_{104}x_{110} \oplus x_{104}x_{109}x_{111} \oplus x_{104}x_{109}x_{110}x_{111} \oplus \\
&x_{104}x_{108}x_{111} \oplus x_{104}x_{108}x_{109}x_{111} \oplus x_{104}x_{108}x_{109}x_{110} \oplus \\
&x_{104}x_{107}x_{110} \oplus x_{104}x_{107}x_{110}x_{111} \oplus x_{104}x_{107}x_{109}x_{111} \oplus \\
&x_{104}x_{107}x_{108}x_{111} \oplus x_{104}x_{107}x_{108}x_{110} \oplus \\
&x_{104}x_{107}x_{108}x_{110}x_{111} \oplus x_{104}x_{107}x_{108}x_{109} \oplus \\
&x_{104}x_{107}x_{108}x_{109}x_{111} \oplus x_{104}x_{106} \oplus x_{104}x_{106}x_{109}x_{110}x_{111} \oplus \\
&x_{104}x_{106}x_{108} \oplus x_{104}x_{106}x_{108}x_{111} \oplus x_{104}x_{106}x_{107} \oplus \\
&x_{104}x_{106}x_{107}x_{110} \oplus x_{104}x_{106}x_{107}x_{110}x_{111} \oplus \\
&x_{104}x_{106}x_{107}x_{109}x_{110}x_{111} \oplus x_{104}x_{106}x_{107}x_{108}x_{110}x_{111} \oplus \\
&x_{104}x_{106}x_{107}x_{108}x_{109}x_{111} \oplus x_{104}x_{105}x_{111} \oplus x_{104}x_{105}x_{109} \oplus \\
&x_{104}x_{105}x_{109}x_{110}x_{111} \oplus x_{104}x_{105}x_{108}x_{111} \oplus \\
&x_{104}x_{105}x_{108}x_{110} \oplus x_{104}x_{105}x_{108}x_{109}x_{110}x_{111} \oplus \\
&x_{104}x_{105}x_{107} \oplus x_{104}x_{105}x_{107}x_{111} \oplus x_{104}x_{105}x_{107}x_{110} \oplus \\
&x_{104}x_{105}x_{107}x_{109} \oplus x_{104}x_{105}x_{107}x_{109}x_{110} \oplus \\
&x_{104}x_{105}x_{107}x_{108}x_{111} \oplus x_{104}x_{105}x_{107}x_{108}x_{110}x_{111} \oplus \\
&x_{104}x_{105}x_{107}x_{108}x_{109}x_{111} \oplus x_{104}x_{105}x_{106}x_{110} \oplus \\
&x_{104}x_{105}x_{106}x_{110}x_{111} \oplus x_{104}x_{105}x_{106}x_{109} \oplus \\
&x_{104}x_{105}x_{106}x_{109}x_{110} \oplus x_{104}x_{105}x_{106}x_{108}x_{111} \oplus \\
&x_{104}x_{105}x_{106}x_{108}x_{110} \oplus x_{104}x_{105}x_{106}x_{108}x_{110}x_{111} \oplus \\
&x_{104}x_{105}x_{106}x_{108}x_{109}x_{111} \oplus x_{104}x_{105}x_{106}x_{107} \oplus \\
&x_{104}x_{105}x_{106}x_{107}x_{110} \oplus x_{104}x_{105}x_{106}x_{107}x_{109}x_{111} \oplus \\
&x_{104}x_{105}x_{106}x_{107}x_{108} \oplus x_{104}x_{105}x_{106}x_{107}x_{108}x_{110} \oplus \\
&x_{104}x_{105}x_{106}x_{107}x_{108}x_{110}x_{111} \oplus \\
&x_{104}x_{105}x_{106}x_{107}x_{108}x_{109}x_{111} \oplus x_0
\end{aligned}
$$

4.3.5 Global Solution

We have now a Boolean function for each function `SubBytes` $SB()$, `ShiftRows` $SR()$ and `MixColumns` $MC()$. In the arrangement of one round, these functions are combined. So for a 128-bit block $B = (b_1, \cdots, b_{128})$ as output of the `AddRoundKey` function, the block $B' = (b'_1, \cdots, b'_{128})$ as output of the combination of these three functions is such that:

$$B' = MC \circ SR \circ SB(B)$$

To realize the files as described above, it is necessary to reduce the composition of these three functions in one Boolean equation. To achieve this, we just have to replace each input variable of a function by the output value of the previous function using the following equation:

$$b'_i = MC(SR(SB(b_i))) \quad \forall i \in (1, \cdots, 128)$$

Finally, we can now describe under the form of Boolean equations the full process of AES encryption.

4.4 The Equations for Deciphering Functions

We will now detail the solution implemented for each of the sub-functions of the AES decryption algorithm.

4.4.1 Solution for the Round Function

The AES deciphering algorithm uses the `InvShiftRows`, `InvSubBytes` and `InvMixColumns` functions. Those functions are respectively the inverse functions of `ShiftRows`, `SubBytes` and `MixColumns` functions, used in the ciphering process. The pseudo code of the decryption function can be written as follows, `Nb` corresponding to the 32-bits words number and `Nr` corresponding to the rounds number used in the algorithm.

```
1: function InvCipher(byte in[4*Nb], byte out[4*Nb],
   word w[Nb*(Nr+1)])
2:     byte state[4,Nb]
3:     state ← in
4:     AddRounkey(state, w[Nr*Nb, (Nr+1)*Nb-1])
5:     for round=Nr-1 step -1 downto 1 do
6:         InvShiftRows(state)
7:         InvSubBytes(state)
8:         AddRoundKey(state, w[round*Nb,
   (round+1)*Nb-1])
9:         InvMixColumns(state)
10:    end for
11:    InvShiftRows(state)
12:    InvSubBytes(state)
13:    AddRounkey(state, w[0, Nb-1])
14:    return state
15: end function
```

The internal mechanisms to the three functions used in the round during decryption are similar to encryption functions. So we use the same reasoning as the one implemented earlier to generate the corresponding Boolean equations.

4.4.2 Solution for the Key Expansion Function

The key expansion function is the same for both ciphering and deciphering process. Boolean equations we built previously are reusable.

4.4.3 Global Solution

We have now a Boolean equation for each of `InvSubBytes` $ISB()$, `InvShiftRows` $ISR()$ and `InvMixColumns` $IMC()$ functions. However, unlike the arrangement of intermediate rounds of the encryption process, these three functions are not combined among them. Indeed, the function `AddRoundKey` no longer occurs at the end of the round but sits between `InvSubBytes` and `InvMixColumns` functions.

Thus, for a block $B = (b_1, \cdots, b_{128})$ and a key $K = (k_1, \cdots, k_{128})$ as input of the round, the block $B' = (b'_1, \cdots, b'_{128})$ as output is such that:

$$B' = IMC(ISB \circ ISR(B) \oplus AD(K))$$

To reduce the Boolean equations, we will not therefore be able to combine the equations of `InvSubBytes` and `InvShiftRows`. As before, to achieve this we just have to replace each input variable of a function with its output value of the previous function using the following equation:

$$b'_i = ISB(ISR(b_i)) \quad \forall i \in (1, \cdots, 128)$$

As for the encryption process, we can now describe under the form of Boolean equations the full process of the AES decryption.

4.5 Implementation and Proof

We now have two systems of Boolean equations corresponding to the encryption process and decryption of AES. These two systems each have:

- 128 equations, one for each bit block;
- 1280 variables for the input block;
- 1280 variables for the key.

Concerning the variables of keys, the fact that we have a Boolean equation by round key involve that we have a set of 128 new variables at each round that is 1280 variables for the AES-128. Each of the variables of the n round key being described in terms of variables of the $n-1$ round key. Consequently and due to the `XOR` bitwise operation between the round key and the bits resulting from the round function, we are obliged to insert a new set of 128 variables to describe the block transformation at each round.

Finally we described the AES encryption and decryption process in the form of two systems of Boolean equations with 128 equations and 2560 variables.

This mechanism allows us then to describe all of the AES encryption process in the form of files using the same representation as described above. So we have 128 files, one by bit of block. In these files, each line describes a monomial and the transition from one line to the next is done by the `XOR` operation.

To implement this mechanism of the description of the AES encryption algorithm and generate the 128 files, we have developed and used a python script based on that described earlier in our presentation of AES[2].

The main program, `aes_equa.py`, offers the possibility of one hand to generate the files for AES ciphering and deciphering functions with the `generateEncFullFiles()` and

[2]The source file is available at the link https://github.com/archoad/BooleanAES. This program requires a working Python environment it is cross-platform and does not use specific libraries.

`generateDecFullFiles()` functions and on the other hand, to control that the encryption and the decryption obtained from files is consistent.

Thus, the functions `controlEncFullFiles()` and `controlDecFullFiles` performs respectively the encryption and the decryption from the previously generated files. The function `controlEncFullFiles()` takes as input a block of 128 bits of plain text and a 128-bit block of key while the function `controlDecFullFiles()` takes as input a block of 128 bits of cipher text and a a 128-bit block of key. The selected blocks are those provided as test vectors in Appendix B of FIPS 197 (NIST, 2001). The obtained results correspond to those provided in the FIPS: files we generated well represent the AES encryption and decryption algorithm.

4.5.1 Results Obtained from the Ciphering Process

The result obtained by the function `generateEncFullFiles()` is shown in appendix 5.1 and the result obtained by the `controlEncFullFiles()` is shown in the appendix 5.2. The control function `controlEncFullFiles()` injects in the Boolean functions the 128 initial variables corresponding to the clear text block and the 1280 variables corresponding to the key blocks of each round.

4.5.2 Results Obtained from the Deciphering Process

According to the same principle as for Boolean functions of encryption, the result obtained by the function `generateDecFullFiles()` is shown in the appendix 5.3 and the obtained result from the `controlDecFullFiles()` function is shown in the appendix 5.4.

In both cases, encryption and decryption, the results we obtain by using our files to cipher and to decipher blocks are conform to those described in the FIPS 197. So our Boolean equation system describing the AES algorithm is right.

5 CONCLUSION

After presenting briefly the Boolean algebra, Boolean functions and two of their presentations, we have developed a process that allows us to translate the AES encryption and decryption algorithms in Boolean functions. Then we defined a mode of representation of these Boolean functions in the form of computer files. Finally, we have developed a program to implement this process and to check that the expected results are consistent with those provided in the FIPS.

In the end, we got a two new systems of Boolean equations, the first one describing the entire ciphering process while the second describes the entire deciphering process of the *Advanced Encryption Standard* and each one including 128 equations and $(128 \times 10) + (128 \times 10) = 2560$ variables.

The next step could be to search, through statistical and combinatorial analysis, new ways to cryptanalyse the AES. Either by finding a solution to resolve our equations system either by using statistical bias exploitable with this system.

REFERENCES

Carlet, C. (2010a). *Boolean Functions for Cryptography and Error Correcting Codes*. Cambridge University Press. Chapter of the monography “Boolean Models and Methods in Mathematics, Computer Science, and Engineering”.

Carlet, C. (2010b). *Vectorial Boolean Functions for Cryptography*. Cambridge University Press. Chapter of the monography “Boolean Models and Methods in Mathematics, Computer Science, and Engineering”.

CNSS (2012). National information assurance policy on the use of public standards for the secure sharing of information among national security systems. https://www.cnss.gov.

Courtois, N. and Pieprzyk, J. (2002). Cryptanalysis of block ciphers with overdefined systems of equations. *Cryptology ePrint Archive, Report 2002/044*. https://eprint.iacr.org/2002/044.pdf.

Daemen, J. and Rijmen, V. (1999). AES proposal: Rijndael. http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf.

Dubois, M. and Filiol, E. (2011). Proposal for a new equation system modelling of block ciphers. *Proceedings of the 2nd IMA Conference on Mathematics in Defence*. http://www.ima.org.uk/db/documents/Dubois.pdf.

Dubois, M. and Filiol, E. (2012a). Proposal for a new equation system modelling of block ciphers and application to AES 128. *Proceedings of the 11th European Conference on Information Warfare and Security*, pages 303–312.

Dubois, M. and Filiol, E. (2012b). Proposal for a new equation system modelling of block ciphers and application to AES 128 - long version. *Pioneer Journal of Algebra, Number Theory and its Applications*, 4:11–40.

McCarty, P. (1986). *Introduction to Arithmetical Functions*. Springer.

Menezes, A., Oorschot, P., and Vanstone, S. (1997). *Handbook of applied cryptography*. CRC Press.

Murphy, S. and Robshaw, M. (2002). Essential algebraic structure within the AES. *Advances in Cryptology - CRYPTO 2002*, 2442:1–16.

NIST (2001). Advanced encryption standard. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

O'Donnel, R. (2014). *Analysis of Boolean Functions*. Cambridge University Press.

APPENDIX

5.1 Result of the Files Creation Program for Encryption

```
./aes_equa.py
## Ciphering process
## Create directory AES_files
## AddRoundKey0
## Round0
## AddRoundKey1
## Round1
## AddRoundKey2
## Round2
## AddRoundKey3
## Round3
## AddRoundKey4
## Round4
## AddRoundKey5
## Round5
## AddRoundKey6
## Round6
## AddRoundKey7
## Round7
## AddRoundKey8
## Round8
## AddRoundKey9
## Round9
## AddRoundKey10
## Files generated
```

5.2 Result of the Files Control Program for Encryption

```
 ./aes_equa.py
## Clear block 00112233445566778899aabbccddeeff
## Key block 000102030405060708090a0b0c0d0e0f
## addRoundKey0
000102030405060708090a0b0c0d0e0f0 32
## Round0
5f72641557f5bc92f7be3b291db9f91a 32
## addRoundKey1
89d810e8855ace682d1843d8cb128fe4 32
## Round1
ff87968431d86a51645151fa773ad009 32
## addRoundKey2
4915598f55e5d7a0daca94fa1f0a63f7 32
## Round2
4c9c1e66f771f0762c3f868e534df256 32
## addRoundKey3
fa636a2825b339c940668a3157244d17 32
## Round3
6385b79ffc538df997be478e7547d691 32
## addRoundKey4
247240236966b3fa6ed2753288425b6c 32
## Round4
f4bcd45432e554d075f1d6c51dd03b3c 32
## addRoundKey5
c81677bc9b7ac93b25027992b0261996 32
## Round5
9816ee7400f87f556b2c049c8e5ad036 32
## addRoundKey6
c62fe109f75eedc3cc79395d84f9cf5d 32
## Round6
c57e1c159a9bd286f05f4be098c63439 32
## addRoundKey7
d1876c0f79c4300ab45594add66ff41f 32
## Round7
baa03de7a1f9b56ed5512cba5f414d23 32
## addRoundKey8
fde3bad205e5d0d73547964ef1fe37f1 32
## Round8
e9f74eec023020f61bf2ccf2353c21c7 32
## addRoundKey9
bd6e7c3df2b5779e0b61216e8b10b689 32
## Round9
7ad5fda789ef4e272bca100b3d9ff59f 32
## addRoundKey10
69c4e0d86a7b0430d8cdb78070b4c55a 32
69c4e0d86a7b0430d8cdb78070b4c55a (FIPS result)
```

5.3 Result of the File Creation Program for Decryption

```
./aes_equa.py
## Deciphering process
## Create directory AES_files
## AddRoundKey10
## Round 9
## AddRoundKey9
## InvMixColumns 9
## Round 8
## AddRoundKey8
## InvMixColumns 8
## Round 7
## AddRoundKey7
## InvMixColumns 7
## Round 6
## AddRoundKey6
## InvMixColumns 6
## Round 5
## AddRoundKey5
## InvMixColumns 5
## Round 4
## AddRoundKey4
## InvMixColumns 4
## Round 3
## AddRoundKey3
## InvMixColumns 3
## Round 2
```

```
## AddRoundKey2
## InvMixColumns 2
## Round 1
## AddRoundKey1
## InvMixColumns 1
## Round 0
## AddRoundKey0
## Files generated
```

5.4 Result of the Files Control Program for Decryption

```
./aes_equa.py
## Cipher block 69c4e0d86a7b0430d8cdb78070b4c55a
## Key block 000102030405060708090a0b0c0d0e0f
## addRoundKey10
7ad5fda789ef4e272bca100b3d9ff59f 32
## Round9
bd6e7c3df2b5779e0b61216e8b10b689 32
## addRoundKey9
e9f74eec023020f61bf2ccf2353c21c7 32
## invMixColumns9
54d990a16ba09ab596bbf40ea111702f 32
## Round8
fde3bad205e5d0d73547964ef1fe37f1 32
## addRoundKey8
baa03de7a1f9b56ed5512cba5f414d23 32
## invMixColumns8
3e1c22c0b6fcbf768da85067f6170495 32
## Round7
...
## Round3
fa636a2825b339c940668a3157244d17 32
## addRoundKey3
4c9c1e66f771f0762c3f868e534df256 32
## invMixColumns3
3bd92268fc74fb735767cbe0c0590e2d 32
## Round2
4915598f55e5d7a0daca94fa1f0a63f7 32
## addRoundKey2
ff87968431d86a51645151fa773ad009 32
## invMixColumns2
a7be1a6997ad739bd8c9ca451f618b61 32
## Round1
89d810e8855ace682d1843d8cb128fe4 32
## addRoundKey1
5f72641557f5bc92f7be3b291db9f91a 32
## invMixColumns1
6353e08c0960e104cd70b751bacad0e7 32
## Round0
00102030405060708090a0b0c0d0e0f0 32
## addRoundKey0
00112233445566778899aabbccddeeff 32
00112233445566778899aabbccddeeff (FIPS result)
```