THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 14, Special Issue 2013, pp. 303–316

TOWARDS AN ALGEBRAIC ATTACK ON AES-128 FASTER THAN BRUTE-FORCE

Andrei SIMION[*], Gabriel NEGARA[**]

[*] Independent researcher, Bucharest, ROMANIA
[**] "Al. I. Cuza" University of Iasi, ROMANIA
Corresponding author: Gabriel NEGARA, E-mail: negara10@gmail.com

In this paper we describe the main ideas of a few versions of an algebraic known plaintext attack against AES-128. The attack could be applied under the hypothesis of knowing a part of the 16-bytes key. These attack versions are based on some specific properties of the key schedule, properties that allow splitting the keys space (2^{128} keys) in subspaces based on some well-defined criteria. The practical efficiency of these attacks depends on some conditions including a conjecture. Also, the paper introduces a definition of weak keys, in the context of the presented attack.

Key words: AES-128, Rijndael, brute-force attack, algebraic attacks, multivariate binary equations system, key schedule, expanded keys.

1. INTRODUCTION

The Advanced Encryption Standard, with its three versions, AES-128, AES-192, and AES-256 is the best known and most widely used block cipher [1]. A detailed description of AES, including some attacks can be found in [2].

In the **2nd Section** of the paper we introduce a scheme, containing the 11 round keys in a compact form which leads to a series of properties of the key schedule, especially the way in which the bytes from the initial key contribute to obtaining the round keys bytes.

Based on these properties, we present some ways in which the 2^{128} keys can be split in classes (subspaces), the computation needed for the expansion of all the keys (at once) within a class being more efficient (reduced) than the computation needed for obtaining the round keys for each key from that specific class.

We also study the case when all the 16 bytes of the initial key K^0 are known, as in the brute-force approaches.

The **3rd Section** presents versions of potential algebraic attacks against AES-128. In these versions 1, 2, 4, or 8 bytes out of 16 of the initial key are considered unknown, while the other 15, 14, 12 or 8 bytes are considered known. It is shown that the 11 round keys can be easily computed even if we consider unknown 8 bytes from the initial key.

The attack versions could be more efficient than a brute-force approach if some conditions are satisfied, including a conjecture.

In this work we present in more detail only the version of the algebraic attack in which just 1 byte from the initial key is considered unknown; for the other possible attack scenarios only the main ideas are presented.

In order to obtain the equations system that needs to be tested for compatibility we use an algorithm for computing S-box's output when providing as input binary vectors.

Improvements of these attacks can be made by taking into account some properties of the S-box and also by using some feasible approaches (like in the case of the key schedule) that could lead to a better structure of the equations systems obtained.

In the **4th Section** we present a definition of keys considered weak in respect to some criteria. For this we expand the key in inverse order, from round 10 to round 0, because in this way is much easier to identify the 'weak keys' in the sense we defined them.

Also, the attack version using the key expanded in inverse order and considering unknown 1 byte of the key K^{10} could be more efficient than the attack considering the key expanded in the usual way and, again, 1 byte of the key unknown.

2. PROPERTIES OF THE KEY SCHEDULE

The initial key K^0 and the 10 round keys K^1, K^2, ..., K^{10} (each containing 16 bytes) are all called "expanded key".

Let us shortly remind the iterative computation of round key K^{i+1} from the round key K^i:

If

$$K^i = \begin{bmatrix} K_{00}^i & K_{10}^i & K_{20}^i & K_{30}^i \\ K_{01}^i & K_{11}^i & \dots & \dots \\ K_{02}^i & \dots & \dots & \dots \\ K_{03}^i & \dots & \dots & \dots \end{bmatrix}$$

then the bytes of the round key K^{i+1} are obtained by:

$$K^{i+1} = \begin{bmatrix} K_{00}^{i+1} = K_{00}^i + SBox(K_{31}^i) + C^i & K_{10}^{i+1} = K_{00}^{i+1} + K_{10}^i & \dots & \dots \\ K_{01}^{i+1} = K_{01}^i + SBox(K_{32}^i) & \dots & \dots & \dots \\ K_{02}^{i+1} = K_{02}^i + SBox(K_{33}^i) & \dots & \dots & \dots \\ K_{03}^{i+1} = K_{03}^i + SBox(K_{30}^i) & \dots & \dots & \dots \end{bmatrix}$$

where + represents the bitwise operation XOR (applied for 2 bytes) and C^i represents the round constant.

It is difficult to notice some properties of the key by describing the expanded key using the above relations.

Using the conventions (for page fitting):

– *xy* meaning *x XOR y*;

– *[x]* meaning *SBox(x);*

and other adequate notations, the expanded key can be written in a compact and easily readable form.

The expanded key is presented in the first 4 columns of Scheme 1, the used notations being written in the 5th column; the values 01, 02, 04, ..., 1B, 36 represent the round constants.

	1	2	3	4	5
0	a	e	i	m	
0	b	f	j	n	
0	c	g	k	o	
0	d	h	l	p	
1	eimA1	imA1	mA1	A1	A1=aeim[n]01
1	fjnB1	jnB1	nB1	B1	B1=bfjn[o]
1	gkoC1	koC1	oC1	C1	C1=cgko[p]
1	hlpD1	lpD1	pD1	D1	D1=dhlp[m]
2	iA1B2	mB2	A1B2	B2	B2=em[B1]02
2	jB1C2	nC2	B1C2	C2	C2=fn[C1]
2	kC1D2	oD2	C1D2	D2	D2=go[D1]
2	lD1A2	pA2	D1A2	A2	A2=hp[A1]
3	mA1B2C3	A1C3	B2C3	C3	C3=im[C2]04
3	nB1C2D3	B1D3	C2D3	D3	D3=jn[D2]
3	oC1D2A3	C1A3	D2A3	A3	A3=ko[A2]
3	pD1A2B3	D1B3	A2B3	B3	B3=lp[B2]

(*continued*)

4	A1B2C3D4	B2D4	C3D4	D4	D4=m[D3]08
4	B1C2D3A4	C2A4	D3A4	A4	A4=n[A3]
4	C1D2A3B4	D2B4	A3B4	B4	B4=o[B3]
4	D1A2B3C4	A2C4	B3C4	C4	C4=p[C3]
5	B2C3D4A5	C3A5	D4A5	A5	A5=A1[A4]10
5	C2D3A4B5	D3B5	A4B5	B5	B5=B1[B4]
5	D2A3B4C5	A3C5	B4C5	C5	C5=C1[C4]
5	A2B3C4D5	B3D5	C4D5	D5	D5=D1[D4]
6	C3D4A5B6	D4B6	A5B6	B6	B6=B2[B5]20
6	D3A4B5C6	A4C6	B5C6	C6	C6=C2[C5]
6	A3B4C5D6	B4D6	C5D6	D6	D6=D2[D5]
6	B3C4D5A6	C4A6	D5A6	A6	A6=A2[A5]
7	D4A5B6C7	A5C7	B6C7	C7	C7=C3[C6]40
7	A4B5C6D7	B5D7	C6D7	D7	D7=D3[D6]
7	B4C5D6A7	C5A7	D6A7	A7	A7=A3[A6]
7	C4D5A6B7	D5B7	A6B7	B7	B7=B3[B6]
8	A5B6C7D8	B6D8	C7D8	D8	D8=D4[D7]80
8	B5C6D7A8	C6A8	D7A8	A8	A8=A4[A7]
8	C5D6A7B8	D6B8	A7B8	B8	B8=B4[B7]
8	D5A6B7C8	A6C8	B7C8	C8	C8=C4[C7]
9	B6C7D8A9	C7A9	D8A9	A9	A9=A5[A8]1B
9	C6D7A8B9	D7B9	A8B9	B9	B9=B5[B8]
9	D6A7B8C9	A7C9	B8C9	C9	C9=C5[C8]
9	A6B7C8D9	B7D9	C8D9	D9	D9=D5[D8]
10	C7D8A9B10	D8B10	A9B10	B10	B10=B6[B9]36
10	D7A8B9C10	A8C10	B9C10	C10	C10=C6[C9]
10	A7B8C9D10	B8D10	C9D10	D10	D10=D6[D9]
10	B7C8D9A10	C8A10	D9A10	A10	A10=A6[A9]

Scheme 1. The expanded key, using some specific notations.

In the scheme above, for example

$$A1=aeim[n]01$$

means

$$A1=a\ XOR\ e\ XOR\ i\ XOR\ m\ XOR\ SBox(n)\ XOR\ 01.$$

Using the notations in column 5 a series of properties of the expanded key can be revealed, especially the way the bytes of the initial key are found as structural parts of the expanded keys.

First, Table 1 indicates the bytes from the initial key K^0 that are components of the expressions of type *A*, *B*, *C* and *D*.

Table 1

Components Bytes in Expressions *A, B, C, D*

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*o*	*p*
A1	*a*				*e*				*i*				*m*	*n*		
A2	*a*				*e*			*h*	*i*				*m*	*n*		*p*
A3-A10	*a*				*e*			*h*	*i*		*k*		*m*	*n*	*o*	*p*
B1		*b*				*f*				*j*				*n*	*o*	
B2		*b*			*e*	*f*				*j*			*m*	*n*	*o*	

Table 1 (continued)

B3-B10		*b*			*e*	*f*				*j*		*l*	*m*	*n*	*o*	*p*
C1			*c*				*g*				*k*				*o*	*p*
C2			*c*			*f*	*g*				*k*			*n*	*o*	*p*
C3-C10			*c*			*f*	*g*		*i*		*k*		*m*	*n*	*o*	*p*
D1				*d*				*h*				*l*	*m*			*p*
D2				*d*			*g*	*h*				*l*	*m*		*o*	*p*
D3-D10				*d*			*g*	*h*		*j*		*l*	*m*	*n*	*o*	*p*

From Table 1 and Scheme 1 one can deduce the properties *P1*, *P2*, *P3*, *P4* and *P5* from Table 2. These properties refer to the number of bytes from the round keys K^1, K^2, ..., K^{10} in which some specific byte or bytes of the initial key K^0 appears.

Table 2

The properties *P1*, *P2*, *P3*, *P4* and *P5*

Property	Bytes from K^0	*K1*	*K2*	*K3*	*K4*	*K5*	*K6*	*K7*	*K8*	*K9*	*K10*
P1	*a*	4	6	8	9	9	9	9	9	9	9
P2	*i*	2	5	8	11	12	12	12	12	12	12
P3	*a, c*	8	12	12	12	12	12	12	12	12	12
P4	*a, i*	4	6	10	12	12	12	12	12	12	12
P5	*a,c,i,k*	8	12	12	12	12	12	12	12	12	12

Let's notice that, due to the symmetry of the expanded key:
– the property *P1* also holds for the bytes *b*, *c* and *d*;
– the property *P2* also holds for the bytes *j*, *k* and *l*;
– the property *P3* also holds for the subset *{b, d}*;
– the property *P4* also holds for the subsets *{b, j}*, *{c, k}* and *{d, l};*
– the property *P5* also holds for the subset *{b, d, j, l}*.
Observation: one can take into consideration the other bytes from the initial key, but their properties are less relevant for our purpose: affecting a number of bytes from the round keys as small as possible.

Property *P6*: the bytes *a, b, c* and *d* pass through the S-Box only individually (never in combination of 2, 3 or 4).

Property *P7*: the bytes *a, b, c, d, i, j, k* and *l* pass through the S-Box only in combination of at most 2 or 3.

We will explain the significance of these properties for our potential attack.

In the following we show that the brute-force approaches could be accelerated, at least in the process of obtaining the expanded keys, and in the 3rd Section we present the main ideas for some versions of an algebraic attack using the enounced properties.

In a naive brute-force attack, the keys are generated and used sequentially; in order to obtain *n* expanded keys, 40 S-Box substitutions and 170 exclusive or operations are performed for *n* times.

As a consequence of the fact that the key schedule process is independent from the enciphering process we can first group the expanded keys in some specific classes and store them using small dimension tables. These tables would contain precomputed bytes or combinations of bytes as components of the expanded keys (for example, from Scheme 1, the byte *A1B2C3B4* can be seen as an exclusive or of 4 components).

There are multiple options for grouping the expanded keys space (2^{128}) in such a way that obtaining a group of n expanded keys is significantly less computationally expensive than the usual way (40 substitution and 170 exclusive or for *n* times).

These possibilities of generating classes of expanded keys depend on the ability to perform operations based on the data presented in Scheme 1.

We present only the main ideas:

– the expressions *A1*, *A2*, ..., *A10* depend only on 9 from the 16 bytes of initial key: *a, e, i, m, n, h, p, k, o*; more than that, these expressions do not explicitly depend on the 9 bytes but only on the combinations *aeim*, *n*, *hp* and *ko*;

– the expressions *A1*, *A2*, ..., *A10*, in combination with the expressions *C1*, *C2*, ..., *C10* depend only on 12 bytes of initial key and, like in the previous case, do not explicitly depend on each byte individually, but only on the combinations *ae*, *im*, *cg, ko, f, h, n* and *p.*

3. ALGEBRAIC ATTACK VERSIONS

All the attack versions we present are variations of the known plaintext brute-force attack. A part of the bytes from the initial key K^0 are considered known, the other bytes being recovered from some binary equations systems. Obviously, like in the case of usual brute-force approach, when the equations system is found to be incompatible, the attack will take into consideration some other values for the bytes considered known.

The practical efficiency of such versions of attacks depends especially on:

– the computational effort needed to generate the equations systems;

– the complexity of solving the system, in case of compatibility;

– the efficiency of the false keys filtering.

Related to the false keys filtering, the following conjecture would represent strong result for the success and efficiency of the attack (if true):

Conjecture: given an over-defined binary equations system that could be compatible or incompatible, it is easier to prove that it is incompatible than to find its solution, when compatible.

This conjecture is true for a binary multivariate equations system, when the number of binary variables is small but, probably, it is hard to decide when the number of variables is larger.

In the following, we present the basic ideas of the attack versions successively considering unknown 1, 2, 4 and 8 bytes from the initial key.

First, we depict the way to obtain the expanded key, starting from the Scheme 1.

The Scheme 2 presents the expanded key obtained when the byte $u = a$ is considered unknown. Similar schemes are obtained when the byte considered unknown is *b*, *c*, or *d*. The expanded key when considering unknown the bytes $u = a$ and $v = c$ is presented in Scheme 3 (similar when the bytes *b* and *d* are unknown). The expanded key in the case of unknown bytes $u = a$, $v = c$, $x = b$ and $y = d$ is presented in Scheme 4.

One can observe that the Scheme 2 can be directly obtained from the Scheme 1:

– the values of the expressions *A1*, *A2*, ..., *A10* from the 5th column are modified appropriately;

– in the columns 1, 2, 3, 4, *A1* becomes *A1P0(u)*, *A2* becomes *A2P1(u)*, *A3* becomes *A3P2(u)*, ..., *A10* becomes *A10P9(u)*.

The way to compute the expressions *P0(u), P1(u)*, *P2(u), ..., P9(u)* is specified above the round key where they appear for the first time.

Similarly, the Scheme 3 can be deduced from the Scheme 2, by modifying the values of the expressions *C1*, *C2*, ..., *C10* from the 5th column and by replacing the expressions *C1*, *C2*, ..., *C10* from the columns 1, 2, 3, 4 with *C1Q0(v)*, *C2Q1(v), ...*, *C10Q9(v).*

Also, the Scheme 4 can be deduced from the Scheme 3.

In the Schemes 2 and 3 one can observe that some of the round keys bytes (in green) do not depend on the bytes considered unknown, in conformity with the properties *P1* and *P3*.

Other possibilities of selecting the unknown bytes are shortly presented, without the corresponding schemes for the expanded keys (these schemes cannot be deduced directly from Scheme 1).

Due to the similarities in the calculus of the expressions of *P, Q, R, S* type we will first present the general methodology of calculus, followed then by the analysis of the particularities specific to each scheme.

One can quickly observe that the expressions like *P1, Q1, R1, S1* can be immediately obtained from the truth tables of *SBox(x+c),* where c is a constant from the set {0, 1, 2, ..., 255}, depending on the bytes values from the initial key, that represent components of the expressions *A1, B1, C1, D1*.

	1	2	3	4	5
0	u	e	i	m	
0	b	f	j	n	
0	c	g	k	o	
0	d	h	l	p	
	P0(u)=u01				
1	eimA1P0(u)	imA1P0(u)	mA1P0(u)	A1P0(u)	A1=eim[n]
1	fjnB1	jnB1	nB1	B1	B1=bfjn[o]
1	gkoC1	koC1	oC1	C1	C1=cgko[p]
1	hlpD1	lpD1	pD1	D1	D1=dhlp[m]
	P1(u)=[A1P0(u)]				
2	iA1B2P0(u)	mB2	A1B2P0(u)	B2	B2=em[B1]02
2	jB1C2	nC2	B1C2	C2	C2=fn[C1]
2	kC1D2	oD2	C1D2	D2	D2=go[D1]
2	lD1A2P1(u)	pA2P1(u)	D1A2P1(u)	A2P1(u)	A2=hp
	P2(u)=[A2P1(u)]				
3	mA1B2C3P0(u)	A1C3P0(u)	B2C3	C3	C3=im[C2]04
3	nB1C2D3	B1D3	C2D3	D3	D3=jn[D2]
3	oC1D2A3P2(u)	C1A3P2(u)	D2A3P2(u)	A3P2(u)	A3=ko
3	pD1A2B3P1(u)	D1B3	A2B3P1(u)	B3	B3=lp[B2]
	P3(u)=[A3P2(u)]				
4	A1B2C3D4P0(u)	B2D4	C3D4	D4	D4=m[D3]08
4	B1C2D3A4P3(u)	C2A4P3(u)	D3A4P3(u)	A4P3(u)	A4=n
4	C1D2A3B4P2(u)	D2B4	A3B4P2(u)	B4	B4=o[B3]
4	D1A2B3C4P1(u)	A2C4P1(u)	B3C4	C4	C4=p[C3]
	P4(u)=[A4P3(u)]P0(u)10				
5	B2C3D4A5P4(u)	C3A5P4(u)	D4A5P4(u)	A5P4(u)	A5=A1
5	C2D3A4B5P3(u)	D3B5	A4B5P3(u)	B5	B5=B1[B4]
5	D2A3B4C5P2(u)	A3C5P2(u)	B4C5	C5	C5=C1[C4]
5	A2B3C4D5P1(u)	B3D5	C4D5	D5	D5=D1[D4]
	P5(u)=[A5P4(u)]P1(u)				
6	C3D4A5B6P4(u)	D4B6	A5B6P4(u)	B6	B6=B2[B5]20
6	D3A4B5C6P3(u)	A4C6P3(u)	B5C6	C6	C6=C2[C5]
6	A3B4C5D6P2(u)	B4D6	C5D6	D6	D6=D2[D5]
6	B3C4D5A6P5(u)	C4A6P5(u)	D5A6P5(u)	A6P5(u)	A6=A2
	P6(u)=[A6P5(u)]P2(u)				
7	D4A5B6C7P4(u)	A5C7P4(u)	B6C7	C7	C7=C3[C6]40
7	A4B5C6D7P3(u)	B5D7	C6D7	D7	D7=D3[D6]
7	B4C5D6A7P6(u)	C5A7P6(u)	D6A7P6(u)	A7P6(u)	A7=A3
7	C4D5A6B7P5(u)	D5B7	A6B7P5(u)	B7	B7=B3[B6]
	P7(u)=[A7P6(u)]P3(u)				
8	A5B6C7D8P4(u)	B6D8	C7D8	D8	D8=D4[D7]80
8	B5C6D7A8P7(u)	C6A8P7(u)	D7A8P7(u)	A8P7(u)	A8=A4
8	C5D6A7B8P6(u)	D6B8	A7B8P6(u)	B8	B8=B4[B7]
8	D5A6B7C8P5(u)	A6C8P5(u)	B7C8	C8	C8=C4[C7]
	P8(u)=[A8P7(u)]P4(u)1B				
9	B6C7D8A9P8(u)	C7A9P8(u)	D8A9P8(u)	A9P8(u)	A9=A5
9	C6D7A8B9P7(u)	D7B9	A8B9P7(u)	B9	B9=B5[B8]
9	D6A7B8C9P6(u)	A7C9P6(u)	B8C9	C9	C9=C5[C8]
9	A6B7C8D9P5(u)	B7D9	C8D9	D9	D9=D5[D8]
	P9(u)=[A9P8(u)]P5(u)				
10	C7D8A9B10P8(u)	D8B10	A9B10P8(u)	B10	B10=B6[B9]36
10	D7A8B9C10P7(u)	A8C10P7(u)	B9C10	C10	C10=C6[C9]
10	A7B8C9D10P6(u)	B8D10	C9D10	D10	D10=D6[D9]
10	B7C8D9A10P9(u)	C8A10P9(u)	D9A10P9(u)	A10P9(u)	A10=A6

Scheme 2. The expanded key - 1 unknown byte of K^o.

	1	2	3	4	5
0	u	e	i	m	
0	b	f	j	n	
0	v	g	k	o	
0	d	h	l	p	
	P0(u)=u01		Q0(v)=v		
1	eimA1P0(u)	imA1P0(u)	mA1P0(u)	A1P0(u)	A1=eim[n]
1	fjnB1	jnB1	nB1	B1	B1=bfjn[o]
1	gkoC1Q0(v)	koC1Q0(v)	oC1Q0(v)	C1Q0(v)	C1=gko[p]
1	hlpD1	lpD1	pD1	D1	D1=dhlp[m]
	P1(u)=[A1P0(u)]		Q1(v)=[C1Q0(v)]		
2	iA1B2P0(u)	mB2	A1B2P0(u)	B2	B2=em[B1]02
2	jB1C2Q1(v)	nC2Q1(v)	B1C2Q1(v)	C2Q1(v)	C2=fn
2	kC1D2Q0(v)	oD2	C1D2Q0(v)	D2	D2=go[D1]
2	lD1A2P1(u)	pA2P1(u)	D1A2P1(u)	A2P1(u)	A2=hp
	P2(u)=[A2P1(u)]		Q2(v)=[C2Q1(v)]04		
3	mA1B2C3P0(u)Q2(v)	A1C3P0(u)Q2(v)	B2C3Q2(v)	C3Q2(v)	C3=im
3	nB1C2D3Q1(v)	B1D3	C2D3Q1(v)	D3	D3=jn[D2]
3	oC1D2A3P2(u)Q0(v)	C1A3P2(u)Q0(v)	D2A3P2(u)	A3P2(u)	A3=ko
3	pD1A2B3P1(u)	D1B3	A2B3P1(u)	B3	B3=lp[B2]
	P3(u)=[A3P2(u)]		Q3(v)=[C3Q2(v)]		
4	A1B2C3D4P0(u)Q2(v)	B2D4	C3D4Q2(v)	D4	D4=m[D3]08
4	B1C2D3A4P3(u)Q1(v)	C2A4P3(u)Q1(v)	D3A4P3(u)	A4P3(u)	A4=n
4	C1D2A3B4P2(u)Q0(v)	D2B4	A3B4P2(u)	B4	B4=o[B3]
4	D1A2B3C4P1(u)Q3(v)	A2C4P1(u)Q3(v)	B3C4Q3(v)	C4Q3(v)	C4=p
	P4(u)=[A4P3(u)]P0(u)10		Q4(v)=[C4Q3(v)]Q0(v)		
5	B2C3D4A5P4(u)Q2(v)	C3A5P4(u)Q2(v)	D4A5P4(u)	A5P4(u)	A5=A1
5	C2D3A4B5P3(u)Q1(v)	D3B5	A4B5P3(u)	B5	B5=B1[B4]
5	D2A3B4C5P2(u)Q4(v)	A3C5P2(u)Q4(v)	B4C5Q4(v)	C5Q4(v)	C5=C1
5	A2B3C4D5P1(u)Q3(v)	B3D5	C4D5Q3(v)	D5	D5=D1[D4]
	P5(u)=[A5P4(u)]P1(u)		Q5(v)=[C5Q4(v)]Q1(v)		
6	C3D4A5B6P4(u)Q2(v)	D4B6	A5B6P4(u)	B6	B6=B2[B5]20
6	D3A4B5C6P3(u)Q5(v)	A4C6P3(u)Q5(v)	B5C6Q5(v)	C6Q5(v)	C6=C2
6	A3B4C5D6P2(u)Q4(v)	B4D6	C5D6Q4(v)	D6	D6=D2[D5]
6	B3C4D5A6P5(u)Q3(v)	C4A6P5(u)Q3(v)	D5A6P5(u)	A6P5(u)	A6=A2
	P6(u)=[A6P5(u)]P2(u)		Q6(v)=[C6Q5(v)]Q2(v)40		
7	D4A5B6C7P4(u)Q6(v)	A5C7P4(u)Q6(v)	B6C7Q6(v)	C7Q6(v)	C7=C3
7	A4B5C6D7P3(u)Q5(v)	B5D7	C6D7Q5(v)	D7	D7=D3[D6]
7	B4C5D6A7P6(u)Q4(v)	C5A7P6(u)Q4(v)	D6A7P6(u)	A7P6(u)	A7=A3
7	C4D5A6B7P5(u)Q3(v)	D5B7	A6B7P5(u)	B7	B7=B3[B6]
	P7(u)=[A7P6(u)]P3(u)		Q7(v)=[C7Q6(v)]Q3(v)		
8	A5B6C7D8P4(u)Q6(v)	B6D8	C7D8Q6(v)	D8	D8=D4[D7]80
8	B5C6D7A8P7(u)Q5(v)	C6A8P7(u)Q5(v)	D7A8P7(u)	A8P7(u)	A8=A4
8	C5D6A7B8P6(u)Q4(v)	D6B8	A7B8P6(u)	B8	B8=B4[B7]
8	D5A6B7C8P5(u)Q7(v)	A6C8P5(u)Q7(v)	B7C8Q7(v)	C8Q7(v)	C8=C4
	P8(u)=[A8P7(u)]P4(u)1B		Q8(v)=[C8Q7(v)]Q4(v)		
9	B6C7D8A9P8(u)Q6(v)	C7A9P8(u)Q6(v)	D8A9P8(u)	A9P8(u)	A9=A5
9	C6D7A8B9P7(u)Q5(v)	D7B9	A8B9P7(u)	B9	B9=B5[B8]
9	D6A7B8C9P6(u)Q8(v)	A7C9P6(u)Q8(v)	B8C9Q8(v)	C9Q8(v)	C9=C5
9	A6B7C8D9P5(u)Q7(v)	B7D9	C8D9Q7(v)	D9	D9=D5[D8]
	P9(u)=[A9P8(u)]P5(u)		Q9(v)=[C9Q8(v)]Q5(v)		
10	C7D8A9B10P8(u)Q6(v)	D8B10	A9B10P8(u)	B10	B10=B6[B9]36
10	D7A8B9C10P7(u)Q9(v)	A8C10P7(u)Q9(v)	B9C10Q9(v)	C10Q9(v)	C10=C6
10	A7B8C9D10P6(u)Q8(v)	B8D10	C9D10Q8(v)	D10	D10=D6[D9]
10	B7C8D9A10P9(u)Q7(v)	C8A10P9(u)Q7(v)	D9A10P9(u)	A10P9(u)	A10=A6

Scheme 3. The expanded key - 2 unknown bytes of K^o.

	1	2
0	u	e
0	x	f
0	v	g
0	y	h
	P0(u)=u01	Q0(v)=v
1	eimA1P0(u)	imA1P0(u)
1	fjnB1R0(x)	jnB1R0(x)
1	gkoC1Q0(v)	koC1Q0(v)
1	hlpD1S0(y)	lpD1S0(y)
	P1(u)=[A1P0(u)]	Q1(v)=[C1Q0(v)]
2	iA1B2P0(u)R1(x)	mB2R1(x)
2	jB1C2Q1(v)R0(x)	nC2Q1(v)
2	kC1D2Q0(v)S1(y)	oD2S1(y)
2	lD1A2P1(u)S0(y)	pA2P1(u)
	P2(u)=[A2P1(u)]	Q2(v)=[C2Q1(v)]04
3	mA1B2C3P0(u)Q2(v)R1(x)	A1C3P0(u)Q2(v)
3	nB1C2D3Q1(v)R0(x)S2(y)	B1D3R0(x)S2(y)
3	oC1D2A3P2(u)Q0(v)S1(y)	C1A3P2(u)Q0(v)
3	pD1A2B3P1(u)R2(x)S0(y)	D1B3R2(x)S0(y)
	P3(u)=[A3P2(u)]	Q3(v)=[C3Q2(v)]
4	A1B2C3D4P0(u)Q2(v)R1(x)S3(y)	B2D4R1(x)S3(y)
4	B1C2D3A4P3(u)Q1(v)R0(x)S2(y)	C2A4P3(u)Q1(v)
4	C1D2A3B4P2(u)Q0(v)R3(x)S1(y)	D2B4R3(x)S1(y)
4	D1A2B3C4P1(u)Q3(v)R2(x)S0(y)	A2C4P1(u)Q3(v)
	P4(u)=[A4P3(u)]P0(u)10	Q4(v)=[C4Q3(v)]Q0(v)
5	B2C3D4A5P4(u)Q2(v)R1(x)S3(y)	C3A5P4(u)Q2(v)
5	C2D3A4B5P3(u)Q1(v)R4(x)S2(y)	D3B5R4(x)S2(y)
5	D2A3B4C5P2(u)Q4(v)R3(x)S1(y)	A3C5P2(u)Q4(v)
5	A2B3C4D5P1(u)Q3(v)R2(x)S4(y)	B3R2(x)D5S4(y)
	P5(u)=[A5P4(u)]P1(u)	Q5(v)=[C5Q4(v)]Q1(v)
6	C3D4A5B6P4(u)Q2(v)R5(x)S3(y)	D4B6R5(x)S3(y)
6	D3A4B5C6P3(u)Q5(v)R4(x)S2(y)	A4C6P3(u)Q5(v)
6	A3B4C5D6P2(u)Q4(v)R3(x)S5(y)	B4D6R3(x)S5(y)
6	B3C4D5A6P5(u)Q3(v)R2(x)S4(y)	C4A6P5(u)Q3(v)
	P6(u)=[A6P5(u)]P2(u)	Q6(v)=[C6Q5(v)]Q2(v)40
7	D4A5B6C7P4(u)Q6(v)R5(x)S3(y)	A5C7P4(u)Q6(v)
7	A4B5C6D7P3(u)Q5(v)R4(x)S6(y)	B5D7R4(x)S6(y)
7	B4C5D6A7P6(u)Q4(v)R3(x)S5(y)	C5A7P6(u)Q4(v)
7	C4D5A6B7P5(u)Q3(v)R6(x)S4(y)	D5B7R6(x)S4(y)
	P7(u)=[A7P6(u)]P3(u)	Q7(v)=[C7Q6(v)]Q3(v)
8	A5B6C7D8P4(u)Q6(v)R5(x)S7(y)	B6D8R5(x)S7(y)
8	B5C6D7A8P7(u)Q5(v)R4(x)S6(y)	C6A8P7(u)Q5(v)
8	C5D6A7B8P6(u)Q4(v)R7(x)S5(y)	D6B8R7(x)S5(y)
8	D5A6B7C8P5(u)Q7(v)R6(x)S4(y)	A6C8P5(u)Q7(v)
	P8(u)=[A8P7(u)]P4(u)1B	Q8(v)=[C8Q7(v)]Q4(v)
9	B6C7D8A9P8(u)Q6(v)R5(x)S7(y)	C7A9P8(u)Q6(v)
9	C6D7A8B9P7(u)Q5(v)R8(x)S6(y)	D7B9R8(x)S6(y)
9	D6A7B8C9P6(u)Q8(v)R7(x)S5(y)	A7C9P6(u)Q8(v)
9	A6B7C8D9P5(u)Q7(v)R6(x)S8(y)	B7D9R6(x)S8(y)
	P9(u)=[A9P8(u)]P5(u)	Q9(v)=[C9Q8(v)]Q5(v)
10	C7D8A9B10P8(u)Q6(v)R9(x)S7(y)	D8B10R9(x)S7(y)
10	D7A8B9C10P7(u)Q9(v)R8(x)S6(y)	A8C10P7(u)Q9(v)
10	A7B8C9D10P6(u)Q8(v)R7(x)S9(y)	B8D10R7(x)S9(y)
10	B7C8D9A10P9(u)Q7(v)R6(x)S8(y)	C8A10P9(u)Q7(v)

Scheme 4 (1/2). The expanded key - 4 unknown bytes of K^o (the first 2 columns).

	3	4	5
0	i	m	
0	j	n	
0	k	o	
0	l	p	
	R0(x)=x	S0(y)=y	
1	mA1P0(u)	A1P0(u)	A1=eim[n]
1	nB1R0(x)	B1R0(x)	B1=bfjn[o]
1	oC1Q0(v)	C1Q0(v)	C1=gko[p]
1	pD1S0(y)	D1S0(y)	D1=hlp[m]
	R1(x)=[B1R0(x)]02	S1(y)=[D1S0(y)]	
2	A1B2P0(u)R1(x)	B2R1(x)	B2=em
2	B1C2Q1(v)R0(x)	C2Q1(v)	C2=fn
2	C1D2Q0(v)S1(y)	D2S1(y)	D2=go
2	D1A2P1(u)S0(y)	A2P1(u)	A2=hp
	R2(x)=[B2R1(x)]	S2(y)=[D2S1(y)]	
3	B2C3Q2(v)R1(x)	C3Q2(v)	C3=im
3	C2D3Q1(v)S2(y)	D3S2(y)	D3=jn
3	D2A3P2(u)S1(y)	A3P2(u)	A3=ko
3	A2B3P1(u)R2(x)	B3R2(x)	B3=lp
	R3(x)=[B3R2(x)]	S3(y)=[D3S2(y)]08	
4	C3D4Q2(v)S3(y)	D4S3(y)	D4=m
4	D3A4P3(u)S2(y)	A4P3(u)	A4=n
4	A3B4P2(u)R3(x)	B4R3(x)	B4=o
4	B3C4Q3(v)R2(x)	C4Q3(v)	C4=p
	R4(x)=[B4R3(x)]R0(x)	S4(S0(y))=[D4S3(y)]S0(y)	
5	D4A5P4(u)S3(y)	A5P4(u)	A5=A1
5	A4B5P3(u)R4(x)	B5R4(x)	B5=B1
5	B4C5Q4(v)R3(x)	C5Q4(v)	C5=C1
5	C4D5Q3(v)S4(y)	D5S4(y)	D5=D1
	R5(x)=[B5R4(x)]R1(x)20	S5(y)=[D5S4(y)]S1(y)	
6	A5B6P4(u)R5(x)	B6R5(x)	B6=B2
6	B5C6Q5(v)R4(x)	C6Q5(v)	C6=C2
6	C5D6Q4(v)S5(y)	D6S5(y)	D6=D2
6	D5A6P5(u)S4(y)	A6P5(u)	A6=A2
	R6(x)=[B6R5(x)]R2(x)	S6(y)=[D6S5(y)]S2(y)	
7	B6C7Q6(v)R5(x)	C7Q6(v)	C7=C3
7	C6D7Q5(v)S6(y)	D7S6(y)	D7=D3
7	D6A7P6(u)S5(y)	A7P6(u)	A7=A3
7	A6B7P5(u)R6(x)	B7R6(x)	B7=B3
	R7(x)=[B7R6(x)]R3(x)	S7(y)=[D7S6(y)]S3(y)80	
8	C7D8Q6(v)S7(y)	D8S7(y)	D8=D4
8	D7A8P7(u)S6(y)	A8P7(u)	A8=A4
8	A7B8P6(u)R7(x)	B8R7(x)	B8=B4
8	B7C8Q7(v)R6(x)	C8Q7(v)	C8=C4
	R8(x)=[B8R7(x)]R4(x)	S8(y)=[D8S7(y)]S4(y)	
9	D8A9P8(u)S7(y)	A9P8(u)	A9=A5
9	A8B9P7(u)R8(x)	B9R8(x)	B9=B5
9	B8C9Q8(v)R7(x)	C9Q8(v)	C9=C5
9	C8D9Q7(v)S8(y)	D9S8(y)	D9=D5
	R9(x)=[B9R8(x)]R5(x)36	S9(y)=[D9S8(y)]S5(y)	
10	A9B10P8(u)R9(x)	B10R9(x)	B10=B6
10	B9C10Q9(v)R8(x)	C10Q9(v)	C10=C6
10	C9D10Q8(v)S9(y)	D10S9(y)	D10=D6
10	D9A10P9(u)S8(y)	A10P9(u)	A10=A6

Scheme 4 (2/2). The expanded key - 4 unknown bytes of K^o (the last 3 columns).

In order to compute the values of the other expressions, the output of the S-Box need to be computed first, when providing also vector expressions as input.

Thus, in the key schedule process as well as in the round enciphering process, computing expressions of the form *Sbox(c + P)* is needed, where *c* is a byte constant, and *P* is formed by 8 vectors. These 8 vectors are of dimension 2^{8k}, where k ∈{1, 2, 4} represents the number of bytes in the K^0 key, and can be expressed using a truth table or an Algebraic Normal Form table.

In order to perform such computation, we successively found **3 different methods**. The 3rd method is at least two times faster than the 2nd method and thousands times faster than the 1st one. This 3rd method uses only the truth tables of *P* and *SBox*.

So, we only need to compute *c+ P* and then to compose the functions *SBox* and *(c+ P)*.

Note. We present the first two methods of computing *Sbox(c + P)* in **Appendix**, as some steps of them could be interesting by themselves (including an algorithm for multiplying multivariate binary polynomials).

Regarding the effective calculus of the expressions of type *P, Q, R, S* we consider that for each scheme out of 3 (2, 3 and 4) one can find precomputation options, at least for a part of these expressions. This can be done by using ideas similar to the ones presented in the end of Section 2.

Also, in the same register, one can precompute other components of the expanded key, depending on the known (considered) bytes from the initial key.

Related to the attack versions in which we consider unknown other combinations of bytes from the initial key than the ones in the Schemes 2, 3 and 4, we can mention that these versions could present some advantages and also disadvantages. For example, considering as unknown the bytes *(a, i, c, k)* or *(b, j, d, l)*, the expanded key contains also some bytes that do not depend on the unknown bytes (due to the *P5* property). Meanwhile, the expanded key obtained in the case of considering as unknown the bytes, *a, b, c, d* do not contain such bytes. But, the unknown bytes *a, b, c, d* do not pass simultaneously through the *S-Box*, computing the involved expressions implying only 2^8-length arrays, while the unknown bytes *a, i, c, k* pass simultaneously through the *S-Box (a* with *i*, *c* with *k)*, for computing the involved expressions being necessary arrays of dimension 2^{16}.

Regarding the algebraic attack version involving 8 unknown bytes *(a, b, c, d, i, j, k, l)*, this could be the most efficient one; the vectorial expressions depend only on combinations of 3 out of 8 unknown values: *{ai, k}*, *{bj, l}*, *{ck, i}* and *{dl, j}*.

Thus, these polynomial expressions could be computed using polynomial expressions of length 2^{24}.

Obtaining the equations systems can be done relatively easily when considering only 1 unknown byte. One could use either the direct key expansion from the Scheme 2, or the inverse expansion from the Scheme 5. In these cases, the length of the binary arrays remains at 256 after passing through the S-Box in the enciphering process. For the *MixColumns* operation, one can use the 32 × 32 binary equivalent matrix; for decreasing the number of operations required, it is recommended for the equations systems computation algorithm to use a meet-in-the-middle strategy.

For the attack versions involving more unknown bytes, the difficulties in computing the equations systems increase due to the simultaneous passing through the S-Box of the unknown bytes, during the enciphering process. When using the vectorial expressions of the multivariate polynomials, the lengths of these vectors could reach infeasible length. For example, for 8 unknown bytes, the length of these vectors could have 2^{64} length.

As a consequence, for these scenarios, new strategies must be used in order to efficiently compute the equations systems, for example by introducing additional variables.

4. INVERSE KEY EXPANSION AND WEAK KEYS

In order to obtain the round key K^i from K^{i-1} the following operations need to be performed (from right to the left):

$$\begin{aligned}
&K_{00}^{i-1} = K_{00}^{i} + SBox(K_{31}^{i-1}) + C^{i-1} \qquad K_{10}^{i-1} = K_{10}^{i} + K_{00}^{i} \qquad \cdots\cdots \qquad K_{30}^{i-1} = K_{30}^{i} + K_{20}^{i}\\
&K_{01}^{i-1} = K_{01}^{i} + SBox(K_{32}^{i-1}) \qquad \cdots \qquad \cdots\cdots \qquad \cdots\\
&K_{02}^{i-1} = K_{02}^{i} + SBox(K_{33}^{i-1}) \qquad \cdots \qquad \cdots\cdots \qquad \cdots\\
&K_{03}^{i-1} = K_{03}^{i} + SBox(K_{30}^{i-1}) \qquad K_{13}^{i-1} = K_{13}^{i} + K_{03}^{i} \qquad \cdots\cdots \qquad K_{33}^{i-1} = K_{33}^{i} + K_{23}^{i}
\end{aligned}$$

	1	2	3	4
0	xr	r	xr	r
0	P3(x)r	P3(x)r	r	r
0	[P1(x)Q1(x)r5]P2(x)r	r	r	r
0	P1(x)Q1(x)r	P1(x)Q1(x)r	P1(x)Q1(x)r	P1(x)Q1(x)r5
1	xr	xr	r	r
1	P3(x)r	r	r	r
1	P2(x)r	P2(x)r	P2(x)r	P2(x)r4
1	P1(x)Q1(x)r	r	P1(x)Q1(x)r	r
	P3(x)=[P2(x)r4]			
2	xr	r	r	r
2	r	r	r	r
2	P2(x)r	R	P2(x)r	r
2	P1(x)Q1(x)r	P1(x)Q1(x)r	r	r
3	xr	xr	xr	xr3
3	r	r	r	r
3	P2(x)r	P2(x)r	r	r
3	P1(x)Q1(x)r	r	r	r
	Q1(x)=[xr3]			
4	xr	r	xr	r
4	r	r	r	r
4	P2(x)r	r	r	r
4	P1(x)r	P1(x)r	P1(x)r	P1(x)r2
	P2(x)=[P1(x)r2]			
5	xr	xr	r	r
5	r	r	r	r
5	r	r	r	r
5	P1(x)r	r	P1(x)r	r
6	xr	r	r	r
6	r	r	r	r
6	r	r	r	r
6	P1(x)r	P1(x)r	r	r
7	xr	xr	xr	xr1
7	r	r	r	r
7	r	r	r	r
7	P1(x)r	r	r	r
	P1(x)=[xr1]			
8	xr	r	xr	r
8	r	r	r	r
8	r	r	r	r
8	r	r	r	r
9	xr	xr	r	r
9	r	r	r	r
9	r	r	r	r
9	r	r	r	r
10	a=x	e=r	i=r	m=r
10	b=r	f=r	j=r	n=r
10	c=r	g=r	k=r	o=r
10	d=r	h=r	l=r	p=r
	r1=eim	r2=hp	r3=eim[n[ko]]40	

Scheme 5. The inverse key expansion - 1 unknown bytes of K^{10}.

For this inverse expansion, from K^{10} to K^0, we did not find yet a compact scheme similar to the direct expansion from Scheme 1. So, in Scheme 5 we present a simplified version, and only for the case in which the only byte considered unknown is K_{00}^{10}.

The unknown byte is called *x* and all the other bytes will be identified by *r*, even their values are not equal. The propagation of the unknown byte was studied and was taken into consideration the fact that the xor operations between the known bytes lead also to known bytes; the same for the S-Box operations.

The naming conventions are the same as in the previous schemes.

One can observe that in Scheme 5 there are 122 known bytes of the expanded key and only 54 unknown bytes, in contrast with the Scheme 1, in which we have only 94 bytes considered known and 82 unknown.

Thus, from the point of view on an algebraic attach version, the Scheme 5 would be more feasible. For this case we did not analyze techniques for precomputation. The Scheme 5 presents another interesting property that leads to a definition of weak keys.

Definition. A key is "weak" in relation with an unknown-considered byte (or a group of bytes) if, for a *relatively large* set of values of the known bytes, the corresponding expanded keys contains less unknown bytes than in the *usual* case.

In our scenario, having the K_{00}^{10} byte unknown, we may consider weak those keys for which *r1 = r3*, because in this case $P_1(x)=Q_1(x)$, determining another 9 bytes of the expanded keys to become known and one more byte, K_{02}^{0}, to have a simpler expression.

After performing a series of operations, we got the relations *r1=eim* and *r3=eim[n[ko]]40*, thus one can consider weak keys those that satisfy the relation *[n[ko]]=40*, with *n, k, o* bytes from the key K^{10} and 40 the constant of round 7.

Similarly, one can find weak keys in relation with K_{01}^{10}, K_{02}^{10} or K_{03}^{10}.

CONCLUSIONS

In this work we present a series of properties of the expanded key; based on these properties and using adequate notations, we obtain several schemes of the expanded key, when considering as known all or part of the bytes from the initial key, while the remaining bytes are considered unknown. The schemes are simple and provide a way to split the expanded keys space in some specific classes. A significant part of these classes' components can be precomputed and stored in tables having reasonable dimensions.

Also, related to the inverse key schedule, we introduced a definition of weak keys.

Using the 3rd method for computing expressions of type *Sbox(c + P)*, in combination with precomputation techniques and an efficient implementation of the *MixColumn* operation (adapted to our approach), we are confident in the existence of an attack on AES-128 faster than naive brute-force.

In consequence, we consider that our study is likely to contribute to the design of such a new attack against AES-128.

FUTURE RESEARCH DIRECTIONS

1. The study of the key schedule, using our approach, for AES-192 and AES-256, looking for some additional properties, like the ones used in the related-key attacks known in the literature [3], [4].

2. The design of more precomputation techniques, for the known bytes and for the polynomials expressions depending on the unknown bytes.

3. The identification of some properties of binary multivariate equations systems for the AES-128 case study that may lead to more efficient filtering of the false keys.

4. The design of a new hybrid attack against AES based on the ideas from this paper, from [5] and using the notion of higher order correlations [6].

APPENDIX

In the first two methods of computing *Sbox(c + P)* we use the well-known algorithm of computing the *Algebraic Normal Form (ANF)* of a Boolean function from the truth table (or vice versa). We name this algorithm „*alg0*".

For a Boolean function

$$f:\{0,1\}^n \rightarrow \{0,1\}^m,$$

if f_T represents the function's truth table, and f_A represents the table of vectors corresponding to the m *ANF* function's components (both tables having dimension $2^n \times m$), it is known that

$$alg0(f_T) = f_A \text{ and } alg0(f_A)=f_T.$$

The 1st method of computing Sbox(c + P).

For this method we use only the *ANF* of the *AES SBox*; *P* represents the 8 vectors corresponding to the 8 polynomials, each having n binary variables.

The $c+P$ operations modifies only the components of the free terms of the 8 vectors. Let $Q=c+P$. In order to perform the operations, we need to replace the variables $x_0, x_1, ..., x_7$ from the *ANF* expressions of the *SBox* by the vectors $Q_0, Q_1, ..., Q_7$. Thus, we need to compute all the „monomials of polynomials" and then to sum them based on the active bits from the *8 ANF* expressions of the *SBox*.

First, we need to perform multiplication operations for some multivariate polynomials of n binary variables.

Let R_{A1} and R_{A2}, respectively R_{T1} and R_{T2}, be the vectors of two multivariate polynomials of n binary variables, corresponding to the *ANF* representations, and respectively, to the truth tables.

We have

$$(1)\ R_{A1} \cdot R_{A2} = alg0(R_{T1}) \cdot alg0(R_{T2}) = alg0(R_{T1} \cdot R_{T2}) = alg0(alg0(R_{A1}) \cdot alg0(R_{A2})).$$

Thus, the multiplication of two multivariate polynomials reduces to two applications of the *alg0* algorithm and the multiplication of the corresponding truth tables (bitwise AND).

This algorithm can be easily extended for the multiplication of *m* multivariate polynomials of *n* binary variables.

The number of operations involved can be reduced by using a $2^k \times 2^k$ table containing the precomputed multiplications of multivariate polynomials of *k* binary variables.

Note*:* this multiplication algorithm was developed by the authors in 2012. Recently, we found a very similar algorithm, published in April, 2013 [7]. The authors claim that their algorithm, „*MultANF*", is new and faster than other algorithms in the literature, for example the one in the software package SAGE (www.sagemath.org).

Because this first method involves a relatively large number of operations (247 multiplications of binary multivariate polynomials, then XOR operations based on the active bits from the 8 *ANF* expressions of the *SBox*) we looked for another method to compute the „monomials of polynomials".

The 2nd method of computing Sbox(c + P).

We first present a more general case. Let f and g be two boolean functions:

$$f:\{0,1\}^p \rightarrow \{0,1\}^n,\ g:\{0,1\}^n \rightarrow \{0,1\}^m$$

and

$$h=g \circ f,\ h:\{0,1\}^p \rightarrow \{0,1\}^m.$$

Let f_A, g_A, h_A, and f_T, g_T, h_T, be the ANF tables and, respectively, the truth tables.

We have $h_A=g_A \circ f_A$ and $h_T=g_T \circ f_T$, resulting that:

$$(2)\ h_A = alg0(h_T) = alg0(g_T \circ f_T) = alg0(alg0(g_A) \circ alg0(f_A)).$$

Thus, the calculus of h_A starting from f_A and g_A reduces to the computation of f_T and h_T using *alg0*, the composition of the two functions in order to obtain h_T and finally to a new application of *alg0* for obtaining h_A.

In order to use this method for computing *SBox(c+P)* in our case, we set:

$$f = c + P,\ g = SBox,\ n = m = 8,\ p = 8k,$$

where $k \in \{1, 2, 4\}$ represents the number of unknown key bytes.

REFERENCES

1. Joan Daemen, Vincent Rijmen, *The Design of Rijndael. AES - The Advanced Encryption Standard,* Springer-Verlag, 2002.
2. Lars R. Knudsen, Matthew J. B. Robshaw, *The Block Cipher Companion,* Springer-Verlag, 2011.
3. Alex Biryukov, Dmitry Khovratovich, *Related-keyCryptanalysis of the full AES-192 and AES-256,* 2009.
4. Alex Biryukov, Orr Dunkelman, Natahn Keller, Dmitry Khovratovich, Adi Shamir, *Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds*, EUROCRYPT, 2010.
5. Nicolas T. Courtois, Josef Pieprzyk, *Cryptnalysis of Block Ciphers with Overdefined Systems of Equations*, eprint.iacr.org/2002/044.pdf, 2002.
6. Andrei Simion, *Contributions to Cryptanalysis of LFSR-based Stream Ciphers*, PhD Thesis, University of Bucharest, Romania, 2008.
7. Subhabrata Samajder, Palash Sarkar, *Fast multiplication of the algebraic normal forms of two Boolean functions,* WCC 2013 - International Workshop on Coding and Cryptography, Bergen, Norway, 2013.

Received June 1, 2013