INFORMATICA, 2015, Vol. 26, No. 1, 51–65

DOI: http://dx.doi.org/10.15388/Informatica.2015.38

An Algorithm for Key-Dependent S-Box Generation in Block Cipher System

Kazys KAZLAUSKAS*, Gytis VAICEKAUSKAS, Robertas SMALIUKAS

Institute of Informatics and Mathematics, Vilnius University
Akademijos 4, LT-08663 Vilnius, Lithuania
e-mail: kazys.kazlauskas@mii.vu.lt, gytis.vaicekauskas@mii.vu.lt, robertas.smaliukas@mii.vu.lt

Received: August 2013; accepted: August 2014

Abstract. A nonlinear substitution operation of bytes is the main strength factor of the Advanced Encryption Standard (AES) and other modern cipher systems. In this paper we have presented a new simple algorithm to generate key-dependent S-boxes and inverse S-boxes for block cipher systems. The quality of this algorithm was tested by using NIST tests, and changing only one bit of the secret key to generate new key-dependent S-boxes. The fact that the S-boxes are key-dependent and unknown is the main strength of the algorithm, since the linear and differential cryptanalysis require known S-boxes. In the second section of the paper, we analyze S-boxes. In the third section we describe the key-dependent S-boxes and inverse S-boxes generation algorithm. Afterwards, we experimentally investigate the quality of the generated key-dependent S-boxes. Comparison results suggest that the key-dependent S-boxes have good performance and can be applied to AES.

Key words: block cipher systems, key-dependent S-boxes, generation algorithm, experimental results.

1. Introduction

The growing number of communication users has led to increasing demand for security measures to protect data transmitted over open channels (Chen *et al.*, 2008; Li *et al.*, 2007; Sakalauskas, 2005). Cryptography is the best method for data protection against active and passive fraud. A cipher system is a set of reversible transformations from the set M of a plaintext into the set C of a cipher text. Each transformation depends on a secret key and the ciphering algorithm. In the block cipher system, the plaintext is divided into blocks and the ciphering is carried out for the whole block (El-Ramly *et al.*, 2001).

Two general principles of block ciphers are confusion and diffusion. Confusion is transformation that changes the dependence of the statistics of the cipher text on the statistics of the plaintext. Diffusion is spreading of the influence of one plaintext bit to many cipher text bits with the intention to hide the statistical structure of the plaintext. In most cipher systems the confusion and diffusion are achieved by means of round repetition. Repeating a single round contributes to the cipher's simplicity (Masuda *et al.*, 2006). Modern

*Corresponding author.

block ciphers consist of four transformations: substitution, permutation, mixing, and key-adding (Schneier, 1996a; Menezes *et al.*, 1997).

Cryptographic objects are private key algorithms, public key algorithms and pseudo-random generators. Block ciphers usually transform the 128 or 256 bits string of the plaintext to a string of the same length ciphertext under control of the secret key. The private key cryptography, such as DES (Data Encryption Standard, 2001), 3DES, and Advanced Encryption Standard (2001) (AES), uses the same key for the sender and for the receiver to encrypt the plaintext and to decrypt the ciphertext. The private key cryptography is more suitable for the encryption of a large amount of data. The public key cryptography, such as the Rivest–Shamir–Adleman (RSA) or Elliptic Curve algorithms, uses different keys for encryption and decryption. The AES algorithm defined by the National Institute of Standards and Technology of the United States has been accepted to replace DES. AES overpasses DES in an improved security because of larger key sizes. AES is suitable for 8 bit microprocessor platforms and 32 bit processors (Su *et al.*, 2003).

Block cipher systems depend on the S-boxes, which are fixed and have no relation with the secret key. So only a changeable parameter is the secret key. The only nonlinear component of AES is S-boxes, so they are an important source of cryptographic strength. Research of the S-box design has focused on determination of S-box properties which yield cryptographically strong ciphers, with the aim of selecting a small number of good S-boxes for use in a block cipher DES and CAST (Menezes *et al.*, 1997). Some results have demonstrated that a randomly chosen S-box of sufficient size will have several of these desirable properties with a high probability (Keliher, 2003). In Mahmoud *et al.* (2013) a dynamic AES-128 with a key-dependent S-box is designed and implemented. The paper of Juremi *et al.* (2012) presents a new AES-like design for key-dependent AES using the S-box rotation method. An approach for designing a key-dependent S-box defined over $GF(2^4)$ in AES is presented in El-Sheikh *et al.* (2012). A key-dependent S-box of AES algorithm using a variable mapping technique is analyzed in Mohammad *et al.* (2009). A key-dependent S-box generation algorithm in AES block cipher system is proposed in the paper of Kazlauskas and Kazlauskas (2009). Hamdy *et al.* (2011) have proposed a customized version of the AES block cipher in which the key-dependent S-box generation algorithm is used. A proposal for key-dependent AES is also analyzed in the paper of Fahmy *et al.* (2005). In the paper of Hosseinkhani *et al.* (2012), the dynamic S-box is generated in the AES cipher system using the secret key. Other systems, using key-dependent S-boxes were proposed in the past, the most well-known of which is Blowfish (Schneier, 1996a, 1996b) and Khufu (Merkle, 1991). Each of these two systems uses the cryptosystem itself to generate the S-boxes.

This paper outlines the work of the authors' investigation into the design of a new pseudo-randomly generated key-dependent S-boxes. Modeling results show, that the proposed algorithm has a good cryptographic strength, with an additional benefit that the algorithm is resistant to the linear and differential cryptanalysis, which requires that the S-boxes be known. In the second section, we briefly analyze substitution S-boxes. The third section of the paper describes proposed key-dependent S-box and inverse S-box generation algorithm. Afterwards, we discuss the experimental results and give conclusions.

2. Substitution S-Boxes

The essential part of every block cipher is an S-box. To secure the cipher against these attacks, the nonlinearity of the S-box should satisfy the maximum input-output correlation, and the difference propagation probability should be minimum.

There are two ways to fight against the linear and differential cryptanalysis. The first one is to create S-boxes with low linear and differential probabilities. The other is to design the round transformation so that only trails with many active S-boxes would occur. The round transformation must be designed in such a way that differential steps with few active S-boxes would be followed by differential steps with many active S-boxes (Masuda *et al.*, 2006).

Substitution is a nonlinear transformation that performs confusion of bits. A nonlinear transformation is essential for every modern encryption algorithm and is proved to be a strong cryptographic primitive against the linear and differential cryptanalysis. Nonlinear transformations are implemented as lookup tables (S-boxes). An S-box with p input bits and q output bits is denoted as $p \to q$. The DES uses eight $6 \to 4$ S-boxes. S-boxes are designed for software implementation on 8-bit processors. The block ciphers with $8 \to 8$ S-boxes are SAFER, SHARK, and AES. For processors with 32-bit or 64-bit words, S-boxes with more output bits provide a high efficiency. The Snefru, Blowfish, CAST, and SQUARE use $8 \to 32$ S-boxes. The S-boxes can be selected at random as in Snefru, and can be computed using a chaotic map, or have some mathematical structure over a finite Galois field. Examples of the last approach are SAFER, SHARK, and AES. S-boxes that depend on the key values are slower, but more secure than the key independent ones (Schneier, 1996b). The use of the key independent chaotic S-boxes are analyzed in Jakimovski and Kocarev (2001), in which the S-box is constructed with a transformation $F((X + K) \bmod M)$, where K is the key (Masuda *et al.*, 2006).

In AES, the S-box generate two transformations in the Galois fields $GF(2)$ and $GF(2^8)$. The S-box is a nonlinear transformation where each byte of the state is replaced by another byte using the substitution table. The first transformation is: an S-box finds the multiplication inverse of the byte in the field $GF(2^8)$. Since it is an algebraic expression, it is possible to mount algebraic attacks. Hence, it is followed by an affine transformation. The affine transformation is chosen in order to make the SubBytes a complex algebraic expression while preserving the nonlinearity property. Both S-box transformations can be expressed in a matrix form as Hsiao et al. (2005, 2006)

$$S' = M \bullet S^{-1} + C, \tag{1}$$

where the sign $\bullet$ is multiplication and the sign $+$ is addition in the field $GF(2^8)$. The 8×1 vector S' denotes the bits of the output byte after the S-box transformations. The inverse S-box transformation can be get by multiplying both sides of Eq. (1) by M^{-1} and it performs the inverse affine transformation followed by the multiplicative inverse in $GF(2^8)$:

$$S^{-1} = M^{-1} \bullet S' + M^{-1} \bullet C. \tag{2}$$

The S-box elements (bytes) b must be invertible, i.e., $invSbox(Sbox(b)) = b$; must have no fixed elements, i.e., $Sbox(b) = b$; must have no complementary fixed elements, i.e., $Sbox(b) = \bar{b}$, where $\bar{b}$ is bitwise complement of b; $Sbox$ must be not self inverse, i.e., $Sbox(b) \neq invSbox(b)$.

The object of this proposal is a cipher using key-dependent S-boxes. The fact that S-boxes are unknown is one of the main strength of the cipher system, since both linear and differential cryptanalysis require the known S-boxes. If the S-boxes are generated from the key in sufficiently random fashion, each S-box has a high probability of being complete, possessing a fairly high nonlinearity. It is not apparent that the pseudorandom nature of the S-boxes introduces any weakness into the system. Ideal randomness of an S-box cannot be achieved. Ideal randomness is not mathematically possible because of the following reasons: the value of all elements in the S-box difference table should be even, since $a \oplus b = b \oplus a$. Since the S-box is bijective, the input difference of 0 will lead to an output difference of 0. So the element, corresponding to row $= 0$ and column $= 0$ in the difference table, will be 2^n and all other elements in row $= 0$ and column $= 0$ will be 0 (Sakthivel, 2001).

3. Algorithm for Generation of Key-Dependent S-Boxes

Input: 1. The secret key $key(i)$, $i = 1, \ldots, l$ is the vector of l integer numbers (bytes) from the interval $[0, 255]$.
2. The initial substitution box $Sbox(i)$, $i = 0, 1, \ldots, 255$ is the vector of different integer numbers (bytes) from the interval $[0, 255]$.

Output: 1. The key-dependent substitution box $Sboxm(i)$, $i = 0, 1, \ldots, 255$ is the vector of the different integer numbers (bytes) from the interval $[0, 255]$.
2. The key-dependent inverse substitution box $invSboxm(i)$, $i = 0, 1, \ldots, 255$ is the vector of different integer numbers (bytes) from the interval $[0, 255]$.

% Compute the initial value j, which depends on all the secret key's values $key(i)$, $i = 1, 2, \ldots, l$ from the interval $[0, 255]$:

$$1{:}\ j \leftarrow \sum_{i=1}^{l} key(i) \bmod 256$$
2: *for all* $i = 0, 1, \ldots, 255$ *do*

% Compute the index j which depends on the values of the initial substitution box S-box and on the values of the secret key:

$$3{:}\ k \leftarrow \big(Sbox(i) + Sbox(j)\big) \bmod l$$
$$4{:}\ j \leftarrow (j + key(k)) \bmod 256$$

% Replace the values $Sbox(i)$ by the values $Sbox(j)$, and the values $Sbox(j)$ by the values $Sbox(i)$:

5: $p \leftarrow Sbox(i)$
6: $Sbox(i) \leftarrow Sbox(j)$
7: $Sbox(j) \leftarrow p$
8: *end for*

% Write the key-dependent substitution box values to *Sboxm*:

9: $Sboxm \leftarrow Sbox$

% Compute the key-dependent inverse substitution box values *invSboxm*:

10: *for all* $i = 0, 1, \ldots, 255$ *do*
11: $invSboxm(Sboxm(i)) \leftarrow i$
12: *end for*

Remarks.

1. The initial substitution box $Sbox_0$ may be the AES substitution table rearranged according to the rows to the 256-size vector or the ordered numbers $0, 1, \ldots, 255$, or these numbers mixed in any order. In all these cases the sender and the receiver must know these boxes.

2. The length l of the secret key must be 16, 24, or 32 bytes as it is in AES, or the secret key may have any length.

There are six cases of applying the proposed algorithm.

Define: φ – the proposed algorithm; f – the AES round keys generation function; F – the AES ciphering algorithm; *key* – the secret key; $Sbox_0$ – the initial substitution box; $Sboxm_j$ – the jth key-dependent substitution box; $ciphertext_j$ – the result of ciphering round j; $ciphertext_n = ciphertext$; rk – the round keys; $rk_0 = key$, rk_j, $j = 1, 2, \ldots, n$ – the jth round key; n – the number of ciphering rounds (in AES $n = 10$, 12, or 14).

Case 1: The algorithm generates a key-dependent substitution box $Sboxm_0$ for the given secret key *key* and for the initial substitution box $Sbox_0$:

$$\begin{aligned} &1.\ Sboxm_0 = \varphi(key, Sbox_0) \\ &2.\ rk = f(key, Sboxm_0) \\ &3.\ ciphertext = F(rk, Sboxm_0) \end{aligned} \tag{3}$$

Case 2: In each ciphering round j we generate and use the substitution box $Sboxm_j$, which depends on the round key rk_j and on the modified key-dependent initial substitution

box $Sboxm_0$:

$$\begin{array}{l} 1.\ Sboxm_0 = \varphi(key, Sbox_0) \\ 2.\ rk = f(key, Sboxm_0) \\ 3.\ \textit{for all}\ j = 1, \ldots, n\ \ \textit{do} \\ \quad Sboxm_j = \varphi(rk_j, Sboxm_0) \\ \quad ciphertext_j = F(rk_j, Sboxm_j) \\ \ \textit{end} \end{array} \tag{4}$$

Case 3: In each ciphering round j we generate and use the key-dependent substitution box $Sboxm_j$, which depends on the secret key *key* and on the key-dependent substitution box $Sboxm_{j-1}$:

$$\begin{array}{l} 1.\ Sboxm_0 = \varphi(key, Sbox_0) \\ 2.\ rk = f(key, Sboxm_0) \\ 3.\ \textit{for all}\ j = 1, \ldots, n\ \textit{do} \\ \quad Sboxm_j = \varphi(key, Sboxm_{j-1}) \\ \quad ciphertext_j = F(rk_j, Sboxm_j) \\ \ \textit{end} \end{array} \tag{5}$$

Case 4: In each ciphering round j we generate and use the key-dependent substitution box $Sboxm_j$, which depends on the round key rk_j and on the key-dependent substitution box $Sboxm_{j-1}$:

$$\begin{array}{l} 1.\ Sboxm_0 = \varphi(key, Sbox_0) \\ 2.\ rk = f(key, Sboxm_0) \\ 3.\ \textit{for all}\ j = 1, \ldots, n\ \textit{do} \\ \quad Sboxm_j = \varphi(rk_j, Sboxm_{j-1}) \\ \quad ciphertext_j = F(rk_j, Sboxm_j) \\ \ \textit{end} \end{array} \tag{6}$$

Case 5: In each ciphering round j we generate and use key-dependent substitution box $Sboxm_j$, which depends on the secret key *key* and on the key-dependent substitution box $Sboxm_{j-1}$. In addition, round keys *rk* depend on the secret key *key* and the key-dependent substitution box $Sboxm_j$:

$$\begin{array}{l} 1.\ Sboxm_0 = \varphi(key, Sbox_0) \\ 2.\ \textit{for all}\ j = 1, \ldots, n\ \textit{do} \\ \quad Sboxm_j = \varphi(key, Sboxm_{j-1}) \\ \quad rk = f(key, Sboxm_j) \\ \quad ciphertext_j = F(rk_j, Sboxm_j) \\ \ \textit{end} \end{array} \tag{7}$$

Case 6: In each ciphering round j we generate and use key-dependent substitution box $Sboxm_j$, which depends on the round key rk_j and on the key-dependent substitution box $Sboxm_{j-1}$. In addition, round keys rk depend on the secret key key and the key-dependent substitution box $Sboxm_j$:

$$\begin{aligned}
&1.\ Sboxm_0 = \varphi(key, Sbox_0)\\
&2.\ rk = f(key, Sboxm_0\\
&3.\ for\ all\ j = 1, \ldots, n\ do\\
&\quad Sboxm_j = \varphi(rk_j, Sboxm_{j-1})\\
&\quad rk = f(key, Sboxm_j)\\
&\quad ciphertext_j = F(rk_j, Sboxm_j)\\
&\quad end
\end{aligned} \tag{8}$$

The substitution boxes $Sbox(i)$, $Sboxm(i)$, and the inverse substitution boxes $invSbox(i)$, $invSboxm(i)$, $i = 0, 1, \ldots, 255$ are 256-size vectors of the different integer numbers (bytes) from the interval [0, 255]. The indexes i of these vectors are the integer numbers (bytes) from the interval [0, 255]. In the encryption process, the indexes i of the vector *Sbox* (or *Sboxm*) are replaced by the corresponding values of the vector *Sbox* (or *Sboxm*). In the decryption process, the indexes of the vector *invSbox* (or *invSboxm*) are replaced by the corresponding values of the vector *invSbox* (or *invSboxm*). In the decryption process we can use the vector *Sbox* (or *Sboxm*). In such case, the values (bytes) of the vector *Sbox* (or *Sboxm*) are replaced by the corresponding indexes (bytes) of the vector *Sbox* (or *Sboxm*). It is not recommended to apply the *Sbox* (or *Sboxm*) in the decryption process, because the values of the *Sbox* (or *Sboxm*) are mixed (not ordered in an increasing order), and it is not easy to find the corresponding byte in the vector *Sbox* (or *Sboxm*).

4. Experimental Results and Discussion

We introduce the independency measure "*ratio*" of the S-box elements x as follows:

$$ratio = std(corr(\tau)), \tag{9}$$

where $corr(\tau)$ is the correlation function of the normalized S-box elements y:

$$y = \frac{x - mean(x)}{std(x)},$$

in which *mean* is an arithmetic mean and *std* is a standard deviation.

In Eq. (9) $corr(0) = 0$. The normalized *ratio* of (9) is:

$$ratio = ratio/(N - 1), \tag{10}$$

where N is the length of the integer numbers. For AES S-box $N = 256$.

The normalized ratio depends on the interval length of the integer numbers. The AES S-box elements are integers from the interval $[0, 255]$. If we analyze only the interval $[0, 255]$ (16×16 S-box), and suppose that the Matlab function *randperm* permutes the integer numbers "ideally", the normalized *ratio* is equal to 0.0443904 ± 0.0032, and can be used as an "ideal" *ratio* for the random 16×16 S-box. So, we use the normalized *ratio* as the independency measure of the integer numbers from the finite interval. Theoretically, in case the interval of the integer numbers is infinitely long, the normalized *ratio* for the independent integer numbers is equal to zero.

Experiment 1. The aim is to verify the proposed S-box and inverse S-box generation algorithm. Consider the 128 bit length secret key in the hexadecimal form:

$$key = \{17, D5, 4C, 30, D6, 68, C2, 38, 49, D9, 22, 5B, 12, 55, 65, 20\}. \tag{11}$$

We have generated the key-dependent S-box *Sboxm* and an inverse key-dependent S-box *invSboxm*, using the algorithm described in the paper (Case 1). The initial S-box was the AES S-box. The key-dependent S-box *Sboxm* is given in Table 1, and the inverse key-dependent S-box *invSboxm* is given in Table 2. The normalized *ratio* of *Sboxm* elements is equal to 0.0432548 and is approximately equal to the "ideal" normalized *ratio* = 0.044390 for independent numbers generated by using the MATLAB function *randperm*. The S-box consists of all possible permuted 256 values from interval $[0, 255]$. During the encryption each byte is replaced by the corresponding byte from Table 1. For example, byte 91 is replaced by byte EB. During the decryption process the byte EB is replaced by byte 91 (see Table 2).

Table 1
Key-dependent S-box *Sboxm* elements in hexadecimal form.
The initial S-box is AES S-box. The key is as in (11).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	28	F9	B4	66	2F	BD	D2	9F	AD	61	0F	CB	E9	12	13	A8
1	65	93	BC	14	97	07	E4	71	7C	F3	E8	48	F1	FA	33	7A
2	58	2D	0A	4D	56	89	9B	06	3F	7B	78	6F	86	4E	40	6A
3	23	19	87	ED	B6	C8	A6	5B	A2	73	1D	9D	20	D1	0B	E6
4	C2	D5	76	44	5C	9A	17	F7	A3	4F	6D	32	47	A4	8F	27
5	29	50	DA	1F	B8	E7	3E	52	5F	B1	BF	1A	30	A1	DC	3A
6	6E	5E	42	91	D8	C3	51	EC	1E	1C	10	C4	9C	41	2B	7D
7	8E	03	11	45	4A	68	22	0E	FE	F8	77	D6	08	D4	CA	62
8	F0	63	83	80	3B	55	AE	DB	E2	FB	D9	15	C5	E5	F2	CC
9	54	EB	16	2C	B7	CD	C9	DD	E0	B0	DE	24	1B	99	39	3C
A	01	AF	8A	36	64	57	7E	26	C0	AC	6C	35	EE	7F	AB	9E
B	E3	37	B9	85	18	70	6B	8D	4B	04	95	8B	72	BB	00	09
C	96	EF	69	0D	FC	90	D0	46	0C	53	82	BA	98	F6	79	74
D	AA	B5	D3	CF	C1	02	FD	E1	F4	05	EA	81	F5	5D	B3	21
E	25	88	49	DF	67	2A	3D	84	92	B2	C7	8C	BE	75	C6	94
F	31	43	5A	59	2E	60	38	FF	A5	CE	D7	A9	A7	A0	4C	34

Table 2
Key-dependent inverse S-box *invSboxm* elements in hexadecimal form.
The initial S-box is AES S-box. The key is as in (11).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	BE	A0	D5	71	B9	D9	27	15	7C	BF	22	3E	C8	C3	77	0A
1	6A	72	0D	0E	13	8B	92	46	B4	31	5B	9C	69	3A	68	53
2	3C	DF	76	30	9B	E0	A7	4F	00	50	E5	6E	93	21	F4	04
3	5C	F0	4B	1E	FF	AB	A3	B1	F6	9E	5F	84	9F	E6	56	28
4	2E	6D	62	F1	43	73	C7	4C	1B	E2	74	B8	FE	23	2D	49
5	51	66	57	C9	90	85	24	A5	20	F3	F2	37	44	DD	61	58
6	F5	09	7F	81	A4	10	03	E4	75	C2	2F	B6	AA	4A	60	2B
7	B5	17	BC	39	CF	ED	42	7A	2A	CE	1F	29	18	6F	A6	AD
8	83	DB	CA	82	E7	B3	2C	32	E1	25	A2	BB	EB	B7	70	4E
9	C5	63	E8	11	EF	BA	C0	14	CC	9D	45	26	6C	3B	AF	07
A	FD	5D	38	48	4D	F8	36	FC	0F	FB	D0	AE	A9	08	86	A1
B	99	59	E9	DE	02	D1	34	94	54	B2	CB	BD	12	05	EC	5A
C	A8	D4	40	65	6B	8C	EE	EA	35	96	7E	0B	8F	95	F9	D3
D	C6	3D	06	D2	7D	41	7B	FA	64	8A	52	87	5E	97	9A	E3
E	98	D7	88	B0	16	8D	3F	55	1A	0C	DA	91	67	33	AC	C1
F	80	1C	8E	19	D8	DC	CD	47	79	01	1D	89	C4	D6	78	F7

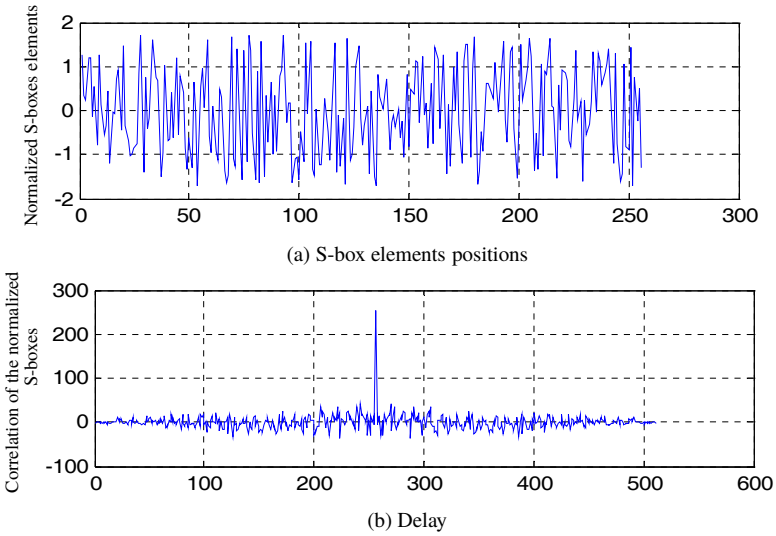
Experiment 2. We have randomly generated 1000 keys and calculated 1000 key-dependent S-boxes *Sboxm*, using the algorithm described in the paper (Case 1). The initial S-box was the AES S-box. The average of the normalized "ratio" of the 1000 key-dependent S-boxes *Sboxm* elements is equal to 0.0440 ± 0.0030, while the normalized "ratio" of the AES S-box elements is 0.0433, i.e., they are approximately equal.

Averaged and normalized elements of 1000 key-dependent S-boxes *Sboxm* via S-box elements positions in the S-box, and averaged and normalized correlation function of 1000 key-dependent S-boxes are given in Fig. 1.

Experiment 3. One of the criteria to evaluate the quality of the block cipher is its suitability as a random number generator. That is, the evaluation of the cipher text using statistical tests should not provide any means by which we could computationally get them from the plaintext.

The following tests are based on NIST Special Publication 800-22rev1a (NIST, 2010), using statistical test suite, called sts-2.1.1. First tested ciphertext was generated using AES (Rijndael) algorithm with standard S-box, hereinafter referred as "Standard algorithm". Other tested ciphertext was generated using AES (Rijndael) algorithm with proposed modification (Case 1), hereinafter referred as "Modified algorithm". The evaluation reported in the paper focused on 128-bit keys. The 160 different types files were encrypted using Standard and Modified algorithms. The encrypted files were used as inputs to the 15 tests of NIST statistical suite. A total of 316408915 128-bit blocks were stored, processed and tested.

Both algorithms used identical plaintexts. This way ciphertext randomness can be checked with recommendations of Special Publication 800-22rev1a and also with each other. In this paper P values of 15 tests are presented, thus letting see which cipher text is more random (under assumption of randomness). The total number of tested sequences



(a) S-box elements positions

(b) Delay

Fig. 1. (a) Averaged and normalized elements of 1000 key-dependent S-boxes *Sboxm* via the positions of S-box elements in the S-box. (b) Averaged and normalized correlation function of the 1000 key-dependent S-boxes. The initial S-box is the AES S-box.

is 160, therefore results are statistically meaningful (NIST, 2010, Section 4.2.2). The averaged P values of the statistical tests for both algorithms are shown in Table 3. Note that both algorithms passed all NIST statistical tests.

Results show that tested ciphertext sequences are random and number of ones is close to the number of ones in a truly random sequence (Frequency test). The numbers of ones and zeros in ciphertext blocks are also close to the numbers of ones and zeros of random sequence (Block Frequency test). Most of other tests also show that tested ciphertext sequences are close to the random sequence (under assumption of randomness).

Some tests show bigger deviation from the results of truly random sequence. However, deeper analysis of all results shows that ciphertexts still are concluded as random. For example, the differences between f_n and Expected Value (L) in the Universal test are smaller than 0.01, therefore ciphertexts are not significantly compressible and this confirms randomness. Delta values of Serial test are not very large and it means that significant uniformity of ciphertexts was not detected.

As the results of the above tests show, both algorithms create ciphertexts that can be considered as random sequences. Comparing P values of ciphertexts of the Standard and Modified algorithms, we can see that more P values are closer to $1/2$ in the ciphertext of Modified algorithm (24 out of 46 P values of successful tests).

Of course, ciphertext strictly depends on the plaintext and the key, because AES algorithm is submissive to avalanche effect and slightly different plaintext will provide very different ciphertext, but that should not have significant impact on current results. Having this in mind we can conclude, that proposed modification of the S-box does not make AES algorithm worse, has no significant influence to the compression or other properties of the ciphertext, therefore it can be used in applications.

Table 3
P values of standard and modified algorithms.

Test	Standard algorithm	Modified algorithm
Frequency	0.52834 ± 0.01432	0.47617 ± 0.01352
Block frequency ($m = 128$)	0.54619 ± 0.01045	0.51347 ± 0.01428
Cusum-forward	0.53145 ± 0.00892	0.50481 ± 0.00943
Cusum-reverse	0.54097 ± 0.09524	0.49964 ± 0.09981
Runs	0.55198 ± 0.01115	0.44960 ± 0.01428
Long runs of ones	0.49216 ± 0.01054	0.50826 ± 0.01083
Rank	0.55835 ± 0.01087	0.47945 ± 0.01204
Spectral DFT	0.53624 ± 0.01427	0.45818 ± 0.01273
Non-overlapping templates ($m = 4$, $B = 0001$)	0.51077 ± 0.01524	0.46769 ± 0.01476
Non-overlapping templates ($m = 4$, $B = 0011$)	0.56063 ± 0.01648	0.45510 ± 0.01376
Non-overlapping templates ($m = 4$, $B = 0111$)	0.44588 ± 0.00845	0.44123 ± 0.00978
Non-overlapping templates ($m = 4$, $B = 1000$)	0.51411 ± 0.01172	0.46740 ± 0.01524
Non-overlapping templates ($m = 4$, $B = 1100$)	0.53061 ± 0.01624	0.46760 ± 0.01045
Non-overlapping templates ($m = 4$, $B = 1110$)	0.44306 ± 0.01092	0.44605 ± 0.01046
Overlapping templates ($m = 9$)	0.51050 ± 0.01167	0.49989 ± 0.01098
Universal	0.37861 ± 0.02045	0.50909 ± 0.01951
Approximate entropy ($m = 10$)	0.46386 ± 0.01842	0.46264 ± 0.01764
Random excursions ($x = -4$)	0.52142 ± 0.02018	0.51764 ± 0.02046
Random excursions ($x = -3$)	0.46244 ± 0.00986	0.43157 ± 0.01059
Random excursions ($x = -2$)	0.50661 ± 0.01056	0.47415 ± 0.01523
Random excursions ($x = -1$)	0.49193 ± 0.01432	0.57688 ± 0.01057
Random excursions ($x = +1$)	0.47158 ± 0.01750	0.52323 ± 0.01645
Random excursions ($x = +2$)	0.49051 ± 0.01341	0.51197 ± 0.01567
Random excursions ($x = +3$)	0.47581 ± 0.01732	0.50464 ± 0.01084
Random excursions ($x = +4$)	0.46657 ± 0.01795	0.44687 ± 0.01345
Random excursions variant ($x = -9$)	0.43187 ± 0.01604	0.45687 ± 0.01548
Random excursions variant ($x = -8$)	0.47602 ± 0.01376	0.45758 ± 0.01273
Random excursions variant ($x = -7$)	0.53423 ± 0.01065	0.51023 ± 0.01270
Random excursions variant ($x = -6$)	0.53684 ± 0.01048	0.51136 ± 0.01486
Random excursions variant ($x = -5$)	0.52567 ± 0.01491	0.44602 ± 0.01507
Random excursions variant ($x = -4$)	0.50906 ± 0.02004	0.43051 ± 0.01976
Random excursions variant ($x = -3$)	0.54154 ± 0.01371	0.41361 ± 0.01504
Random excursions variant ($x = -2$)	0.49632 ± 0.01585	0.40622 ± 0.01349
Random excursions variant ($x = -1$)	0.46422 ± 0.01767	0.46862 ± 0.01465
Random excursions variant ($x = +1$)	0.43194 ± 0.01249	0.48396 ± 0.01182
Random excursions variant ($x = +2$)	0.40270 ± 0.01582	0.44936 ± 0.01620
Random excursions variant ($x = +3$)	0.46795 ± 0.01274	0.45846 ± 0.01473
Random excursions variant ($x = +4$)	0.46380 ± 0.01433	0.47897 ± 0.01627
Random excursions variant ($x = +5$)	0.46886 ± 0.01143	0.49087 ± 0.01842
Random excursions variant ($x = +6$)	0.46846 ± 0.01345	0.49787 ± 0.01167
Random excursions variant ($x = +7$)	0.49613 ± 0.01511	0.48448 ± 0.01279
Random excursions variant ($x = +8$)	0.55272 ± 0.01045	0.49668 ± 0.01192
Random excursions variant ($x = +9$)	0.54552 ± 0.01246	0.51593 ± 0.01164
Linear complexity ($M = 500$)	0.49691 ± 0.01281	0.51203 ± 0.01179
Serial ($m = 16$, $\nabla\Psi_m^2$)	0.27827 ± 0.00974	0.17501 ± 0.00892
Serial ($m = 16$, $\nabla^2\Psi_m^2$)	0.36925 ± 0.00835	0.28943 ± 0.00904

Experiment 4.

1. For randomly generated 1000 keys we have calculated 1000 key-dependent S-boxes *Sboxm* (Case 1). The probability that the value in any key-dependent S-box *Sboxm* position will be equal to the value in the same position of the AES S-box is 0.004. On average, 1.017 values are equal in the same positions of the AES S-box and key-dependent 1000 S-boxes *Sboxm*.
2. To generate S-boxes, we have used key (11), and the initial S-box (AES S-box). Then by randomly changing one bit of the key, we have generated 1000 key-dependent S-boxes *Sboxm*, and calculated equal elements in the same positions of these S-boxes. On average 3 values are equal in the same positions of the AES S-box and the key-dependent 1000 S-boxes.
3. The average correlation coefficient between the AES S-box and 1000 key-dependent S-boxes is 0.0443, i.e., key-dependent S-boxes elements, generated according to the proposed algorithm, practically do not correlate with the AES S-box elements. It follows that the key-dependent S-box generation process is sufficiently random.
4. The avalanche effect property is an important property of an encryption algorithm. It was established that avalanche effect due to one bit change in plaintext is $(50 \pm 3.42)\%$, which means that it is difficult to make predictions on the plaintext being given only the ciphertext. The AES and proposed algorithm have nearly the same avalanche effect.
5. Due to one bit change in the secret key, the avalanche effect of the proposed algorithm is $(50 \pm 3.62)\%$, which means that it is difficult to predict on the plaintext given only the ciphertext. That reflects the immunity of the proposed algorithm to linear and differential cryptanalysis.
6. The calculation time of 1000 key-dependent S-boxes *Sboxm* and 1000 key-dependent inverse S-boxes *invSboxm* with computer AMD Athlon(tm) 64×2 Dual Core Processor 2.59 GHz, 1.87 Gb of RAM is 0.11 s, i.e., the additional time is not significant.

When a permutation uses all elements of the substitution table the permutation $perm(256, 256)$ indicates the number of arrangements that can be formed by selecting 256 elements from a set of 256 elements. In that case $perm(256, 256)$ is equal to the product of integers from 256 to 1 (Korn and Korn, 1961)

$$perm(256, 256) = 256 * 255 * \cdots * 3 * 2 * 1 = 256! \tag{12}$$

So the probability that the cryptanalyst might restore the initial S-box is equal to 1/256! Using our algorithm, we can get up to 256! different substitution values instead of 256 values as in AES algorithm which increase the encryption complexity and complicate the cryptanalysis process. The results show that this is achieved with negligible extra delay.

5. Conclusion

One of the criteria to evaluate the quality of the block cipher is its suitability as a random number generator. That is, the evaluation of the ciphertext using statistical tests should not provide any means by which we could computationally get them from the plaintext. Modeling results show that tested ciphertext sequences are random and number of ones is close to the number of ones in a truly random sequence (Frequency test). The numbers of ones and zeros in ciphertext blocks are also close to the number of random sequence (Block Frequency test). Most of other tests also show that tested ciphertext sequences are close to the random sequence (under assumption of randomness). It has been established that for any change of the secret key, the structure of the key-dependent S-box will be changed essentially. The key-dependent S-boxes make block cipher algorithm resistant to linear and differential cryptanalysis. This algorithm will help to generate more secure block ciphers and will increase the security of the block cipher systems. The main advantage of such algorithm is that the huge number of S-boxes can be generated by changing secret key. The additional time needed to generate key-dependent S-boxes is negligible. As compared with algorithm in the paper Kazlauskas and Kazlauskas (2009), this algorithm generates S-boxes about eight times faster.

Acknowledgements. The authors are gratefully acknowledge the helpful comments of the reviewers.

References

Advanced Encryption Standard (AES) (2001). *Federal Information Processing Standards Publication 197, November 26*.

Chen, T.H., Horng, G., Yang, Ch.S. (2008). Public key autentification schemes for local area networks. *Informatica*, 19(1), 3–16.

Data Encryption Standard (DES) (1977). *National Bureau of Standards FIPS Publication 46*.

El-Ramly, S.H., El-Garf, T., Soliman, A.H. (2001). Dynamic generation of S-boxes in block cipher systems. In: *Eighteenth National Radio Science Conference*, March 27–29, Mansoura University, Egypt, pp. 389–397.

El-Sheikh, H.M., El-Mohsen, O.A., Zekry, A. (2012). A new approach for designing key- dependent S-box defined over GF in AES. *International Journal of Computer Theory and Engineering*, 4(2), 158–164.

Fahmy, A., Shaarawy, M., El-Hadad, K., Salama, G., Hassanain, K. (2005). A proposal for a key-dependent AES. In: *3th International Conference: Sciences of Electronic, Technologies of Information and Telecommunications*, March 27–31, Tunisia.

Hamdy, N., Shehata, K., Eldemerdash, H. (2011). Design and implementation of encryption unit based on customized AES algorithm. *International Journal of Video & Image Processing and Network Security*, 11(1), 33–40.

Hosseinkhani, R., Javadi, H.S. (2012). Using cipher key to generate dynamic S-box in AES cipher system. *International Journal of Computer Science and Security*, 6(1), 19–28.

Hsiao, S.F., Chen, M.C., Tsai, M.Y., Lin, C.C. (2005). System on chip implementation of the whole advanced encryption standard processor using reduced XOR-based sum-of-product operations. *IEEE Proceedings, Information Security*, 152(1), 21–30.

Hsiao, S.F., Chen, M.C., Tu, C.S. (2006). Memory-free low-cost designs of advanced encryption standard using common subexpression elimination for subfunctions in transformations. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 53(3), 615–626.

Jakimovski, G., Kocarev, L. (2001). Chaos and cryptography: block encryption ciphers based on chaotic maps. *IEEE Transactions on Circuits and Systems, Part I*, 48(2), 163–169.

Juremi, J., Mahmod, R., Sulaiman, S., Ramli, J. (2012). Enhancing advanced encryption standard S-box generation based on round key. *International Journal of Cyber-Security and Digital Forensics*, 1(3), 183–188.

Kazlauskas, K., Kazlauskas, J. (2009). Key-dependent S-box generation in AES block cipher system. *Informatica*, 20(1), 23–34.

Keliher, L. (2003). Linear cryptanalysis of substitution-permutation networks. *PhD thesis*, Queen's University, Kingston, Canada.

Korn, G.A., Korn, T.M. (1961). *Mathematical Handbook for Scientists and Engineers*. McGraw-Hill, New York/Toronto/London.

Li, Ch.M., Hwang, T., Lee, N.Y. (2007). Security flow in simple generalized group-oriented cryptosystem using ElGamal cryptosystem. *Informatica*, 18(1), 61–66.

Mahmoud, E.M., El Hafez, A.B., Elgarf, T.A., Zekry, A. (2013). Dynamic AES-128 with key-dependent S-box. *International Journal of Engineering Research and Applications*, 3(1), 1662–1670.

Masuda, N., Jakimovski, G., Aihara, K., Kocarev, L. (2006). Chaotic block ciphers: from theory to practical algorithms. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 53(6), 1341–1352.

Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A. (1997). *Handbook of Applied Cryptography*. CRC Press, Boca Raton.

Merkle, R. (1991). Fast software encryption functions. In: *Advances in Cryptology: Proceedings of CRYPTO'90*. Springer-Verlag, Berlin, pp. 476–501.

Mohammad, F. Y., Rohiem, A.E., Elbayoumy, A.D. (2009). A novel S-box of AES algorithm using variable mapping technique. In: *13th International Conference on Aerospace Sciences & Aviation Technology*, May 2009, Cairo, Egypt, 1/9–10/9.

NIST Special Publication 800-22 revision 1a (April 2010). Technology Administration, US Department of Commerce.

Sakalauskas, E. (2005). On digital signature scheme in semimodule over semiring. *Informatica*, 16(3), 383–394.

Sakthivel, G. (2001). Differential cryptanalysis of substitution permutation networks and Rijndael-like ciphers. *Master's project report*, Rochester Institute of Technology.

Schneier, B. (1996a). *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Wiley, New York.

Schneier, B. (1996b). Description of a new variable-length key, 64-bit block cipher. In: *Proceedings of Fast Software Encryption: Second International Workshop, Leuven, Belgium, December 1994*. Springer-Verlag, pp. 191–204.

Su, C.P., Lin, T.F., Huang, C.T., Wu, C.W. (2003). A high-throughput low-cost AES processor. *IEEE Communications Magazine*, 41(12), 86–91.

K. Kazlauskas received a PhD degree from Kaunas Polytechnic Institute and a doctor habilitus degree from Institute of Mathematics and Informatics and Vytautas Magnus University. He is principal researcher of the Recognition Process Department at the Vilnius University Institute of Mathematics and Informatics, and a professor at the Informatics Department of Lithuanian University of Educational Sciences. His research interests include applied cryptography and digital signal processing.

G. Vaicekauskas received a MSc degree of Informatics from Lithuanian University of Educational Sciences in 2013. Now he is a doctoral student at the Process Recognition Department of the Institute of Mathematics and Informatics, Vilnius University, Lithuania. His research interest is applied cryptography.

R. Smaliukas received a MSc degree of Informatics from Lithuanian University of Educational Sciences in 2014. Now he is a doctoral student at the Process Recognition Department of the Institute of Mathematics and Informatics, Vilnius University, Lithuania. His research interest is applied cryptography.

Blokinių šifravimo sistemų S-lentelių priklausomų nuo rakto generavimo algoritmas

Kazys KAZLAUSKAS, Gytis VAICEKAUSKAS, Robertas SMALIUKAS

AES ir kitų šiuolaikinių šifravimo algoritmų atsparumas labai priklauso nuo netiesinės baitų pakeitimo operacijos. Pasiūlytas naujas paprastas algoritmas, generuojantis priklausomas nuo slaptojo rakto baitų pakeitimo lenteles (S-lenteles), skirtas blokinėms šifravimo sistemoms. Algoritmo kokybė patikrinta naudojant NIST testus, keičiant vieną slaptojo rakto bitą. Straipsnio antrame poskyryje analizuojamos S-lentelės, trečiame – aprašomas algoritmas, o ketvirtame – eksperimentiškai tikrinama sugeneruotų lentelių kokybė.