Recibido 14 Sep 2017 *ReCIBE, Año 6 No. 2, Noviembre 2017*
Aceptado 5 Oct 2017

New S-box calculation approach for Rijndael-AES based on an artificial neural network

Nuevo enfoque para el calculo de la Caja-S para Rijndael-AES basado en una red neuronal artificial

Jaime David Rios Arrañaga[1]
jaime.rios.1xyz@gmail.com

Janneth Alejandra Salamanca Chavarin[1],
salecita_ale@hotmail.com

Juan José Raygoza Panduro[1]
juan.raygoza@cucei.udg.mx

Edwin Christian Becerra Alvarez[1]
edwincbecerra@gmail.com

[1]Centro Universitario de Ciencias Exactas e Ingenierías, Universidad de Guadalajara, Jalisco, México.

Abstract: The S-box is a basic important component in symmetric key encryption, used in block ciphers to confuse or hide the relationship between the plaintext and the ciphertext. In this paper a way to develop the transformation of an input of the S-box specified in AES encryption system through an artificial neural network and the multiplicative inverse in Galois Field is presented. With this implementation more security is achieved since the values of the S-box remain hidden and the inverse table serves as a distractor since it would appear to be the complete S-box. This is implemented on MATLAB and HSPICE using a network of perceptron neurons with a hidden layer and null error.

Keywords: Artificial Neural Network, Cryptography, Circuits, SPICE.

Resumen: La Caja-S es un componente básico en el cifrado de clave simétrica, usado en los cifradores por bloques para confundir o esconder la relación entre el texto plano y el texto cifrado. Este trabajo presenta una manera de desarrollar la transformación de los valores de entrada de la Caja-S especificada en el sistema de cifrado AES por medio de una red neuronal y los valores del inverso multiplicativo en el campo de Galois. Con esta implementación se logra mayor seguridad debido a que los valores de la Caja-S permanecen ocultos mientras que la tabla de los valores inversos en el dominio de Galois sirve de distractor pareciendo ser la verdadera Caja-s. Este trabajo fue implementado en MATLAB y HSPICE utilizando una red con neuronas del tipo Perceptron con una capa oculta, obteniendo los valores esperados por la Caja-S original sin error.

Palabras clave: Circuitos, Criptografia, Red Neuronal Artificial, SPICE

1. Introduction

In cryptography, an S-box consists of a look up table with the corresponding 8-bit word for each possible input in a non-linear transformation, in which the input byte is considered the address of the table (Rodriguez-Henriquez, Saqib, Díaz & Koc 2007). The S-box represents a bricklayer non-linear function that can be decomposed in several boolean functions operating independently on a subset of bits from the input vector (Daemen & Rijmen, 2002). If the functions are linear they are called D-boxes.

The operation of an S-box is as follows: when a transformation is required for a certain input, this input enters the S-box and points, or directs to the previously calculated output of its transformation and then the input is replaced, as shown in fig.1, where the value $a_{i,j}$ is substituted for the value $b_{i,j}$ as it passes through the S-box.

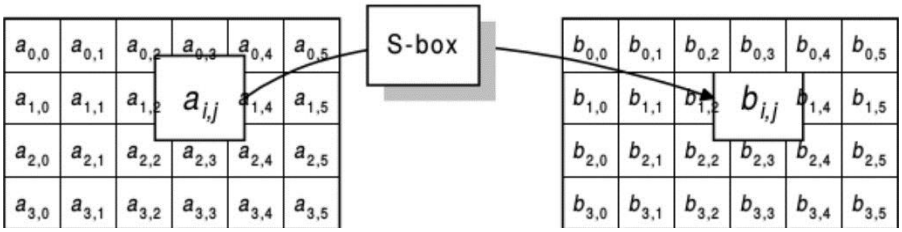

Figure 1. *Graphic representation of the use of an S-box*

Due to their importance, S-boxes are chosen and designed to be resistant to cryptanalysis, in literature several proposals with different characteristics are found, some of them based on neural networks, like the framework for the design of S-boxes used in ciphers based on neural networks by Noughabi (Noughabi & Sadeghiyan, 2010) and “a new scheme for implementing s-box based on neural network” by X. Zhang (Zhang, Chen, Chen, & Cao, 2015), others that optimize existing boxes such as the high speed implementation of S. Oukili for the AES S-box (Oukili, Bri & Kumar, 2016) and low-area S-box implementation of Thomson (Thomson, Siva, & Priya, 2014); even new proposals such as the evolutionary design of S-Box of M. Yang (Yang, Wang, Meng & Han, 2011) and the based on chaotics maps of C. I. Rîncu (Rîncu & Iana, 2014).

This article presents a substitution of the S-box for another module that calculates the AES S-box outputs with the use of a neural network and the multiplicative inverse on Galois field 2^8 (GF (2^8)) of the input value to transform, or S-box input value.

Section 2 introduces the AES algorithm giving a brief introduction to history and a complete description of the Rijndael-AES algorithm, in this section under the subsection “The Round Transformation” highlights the sub-Bytes function that

describe how the values of the S-box are calculated. Section 3 describes the proposed method, this includes the neural network topology and the approach for hardware implementation. The simulations are presented in section 4, this section is an explanation of the implementation, behavior and results in MATLAB and HSPICE. Finally conclusions are given in section 5.

2. AES, Advance Encryption Standard

Developed by Joan Daemen and Vincent Rijimen, Rijndael was finally chosen on October 2000 by the National Institute of Standards and Technology (NIST) among other encryption algorithms in an open process organized by the same institute on January 1997 to become the new *Advanced Encryption Standard* (AES) to replace *Data Encryption Standard* (DES) and triple-DES as encryption standard (Daemen & Rijmen, 2002). Following NIST specifications, AES is a symmetric block cipher algorithm with variable length of 128 bits, 192 bits and 256 bits, with a variable length key of 128 bits, 192 bits y 256 bits and easy on hardware and software implementation (Daemen & Rijmen, 2002).

Although it is common to talk about AES and Rijndael indistinctly, being Rijndael the selected algorithm for AES, there is a difference among them in the range of values supported by the block length and key length to use. In Rijndael, the block length and key length can be independently specified to any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits. AES fixes the length block and the length key to 128, 192 o 256 bits only (Daemen & Rijmen, 1999).

Independently of technical differences in the length of block and key permitted, when talking about Rijndael or AES, we are talking about the same iterative block cipher algorithm. Inputs and outputs of Rijndael-AES are considered to be one-dimensional arrays of 8-bits. For encryption the input is a Plaintext block and a cipher key, and the output is a ciphertext block. For decryption the inputs is a ciphertext block and a cipher key, and the output is a Plaintext block (Daemen & Rijmen, 2002).

The cipher can be divided in two parts with different functionality: the transformation or encoding of the message, function called “The Round transformation” and denoted as “Round” and “FinalRound”, this encryption function is described in fig. 2 along with the functions that make it up, called steps; and the transformation of the key called “Key schedule” given by the function “KeyExpansion”.

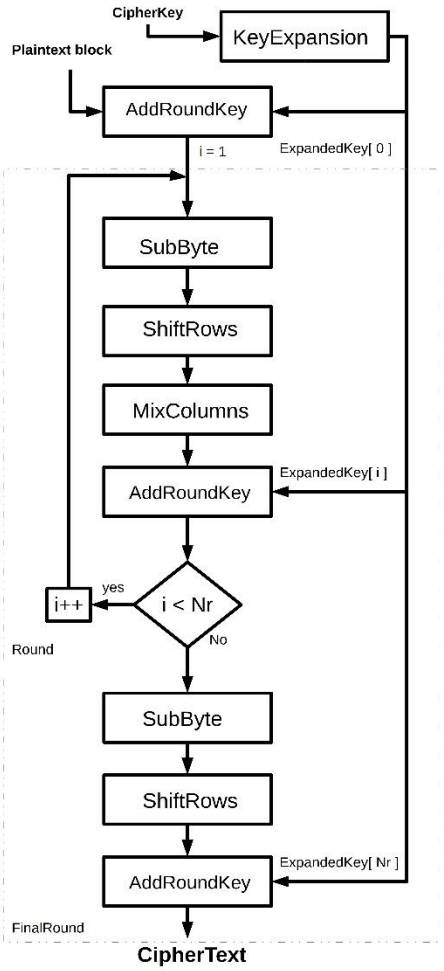

Figure 2. *Flowchart of the AES encryption algorithm*

The different transformation operates on an intermediate result called State which is represented as a rectangular array of bytes, with four rows and N_b number of columns.

$$N_b = \frac{blocklength}{32} \quad (1)$$

Similarly, the cipher key is represented as a rectangular array with four rows and N_k number of columns (Daemen & Rijmen, 1999), (Rodriguez-Henriquez et al., 2007), (Daemen & Rijmen, 2002), (Katz & Lindell, 2008), where

$$N_k = \frac{keylength}{32} \quad (2)$$

The number of rounds N_r depends on the values of N_b and N_k as presented in the table 1.

N_k	Nb				
	4	5	6	7	8
4	10	11	12	13	14
5	11	11	12	13	14
6	12	12	12	13	14
7	13	13	13	13	14
8	14	14	14	14	14

Table 1. *Number Of Rounds N_r As Function Of N_b And N_k*

2.1. The Round Transformation

As shown in the fig. 2, the round transformation is divided in Round and FinalRound. Round is formed by a sequence of four different and invertible mathematical transformations on GF(2^8) which are called steps: *1) SubBytes, 2) ShiftRows, 3) MixColumn, 4) AddRoundKey* (Daemen & Rijmen, 1999), (Rodriguez-Henriquez et al., 2007), (Daemen & Rijmen, 2002). The FinalRound is similar to round but without the MixColumns function.

2.1.1. subBytes.

It is a non-linear transformation where each input byte of the state matrix is replaced by another byte produced by the transformation. This Transformation is defined in two steps (Daemen & Rijmen, 1999):

- Multiplicative inverse:
  The input byte a is replaced by its multiplicative inverse $x = a^{-1}$, with x = 0 for a = 0.
- Affine transformation:
  Defined by $y = M \times x \oplus b$, where M is a constant matrix of 8 × 8 bits, x represents the value to transform while b is a constant byte equal to 63_{16} (01100011_2) (Daemen & Rijmen, 2002).

The matrix representation of the transformation is shown in (3), where M is replaced by the constant matrix of 8×8 bits, x is expanded to the polynomial representation of a byte, starting with the most significant bit; and b the binary constant.

$$\begin{bmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad (3)$$

Another way to implement this transformation is to use the corresponding S-Box shown in fig. 3 replacing the input value (row, column) by the value that crosses them.

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

Figure 3. *AES S-box*

The inverse operation, called InvSubBytes, consists of the use of the inverse S-Box of fig. 4 for each byte of the state.

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

Figure 4. *AES Inverse S-box*

The inverse S-box is obtained by the applying the inverse of the affine transformation, shown in ec. 3 followed by taking the multiplicative inverse in GF(2^8). The inverse of (3) is represented in (4) (Daemen & Rijmen, 2002).

$$\begin{bmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad (4)$$

2.1.2. ShiftRows

In ShiftRows, the rows of the state are shifted cyclically to the left in different proportions. Row 0 does not changes, but the remaining rows follow an offset of C_1 , C_2 and C_3 bytes respectively, this proportion depends only of the block length N_b (Daemen & Rijmen, 2002). The inverse operation, called InvShiftRows, consists in a cyclic shift of the three bottom rows over $N_b - C_1$, $N_b - C_2$ *y* $N_b - C_3$ bytes respectively. The table 2 shows the value of C_n per each possible N_b.

N_b	C_0	C_1	C_2	C_3
4	0	1	2	3
5	0	1	2	3
6	0	1	2	3
7	0	1	2	3
8	0	1	2	3

Table 2. *Shifted Bytes In Shiftrows Per Block Lenght*

2.1.3. MixColumns

The MixColumns step is a bricklayer permutation operating on the state column by column. In Mixcolumns the state columns are considered as polynomials in GF (2^8) and multiplied modulo $x_4 + 1$ with the fixed polynomial c(x) given by *c(x) = (03_{16})x_3 + (01_{16})x_2 + (01_{16})x + 02_{16}* . This operation can be written as a matrix multiplication, let *b(x) = c(x) a(x) mod x^4 +1 as is show in (5).*

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02_{16} & 03_{16} & 01_{16} & 01_{16} \\ 01_{16} & 02_{16} & 03_{16} & 01_{16} \\ 01_{16} & 01_{16} & 02_{16} & 03_{16} \\ 03_{16} & 01_{16} & 01_{16} & 02_{16} \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad (5)$$

The inverse of MixColumns is called InvMixColumns. It is similar to MixColumns.

The transformation is performed by multiplying each column by the polynomial $d(x) = (0B_{16})x_3 + (0D_{16})x_2 + (09_{16})x + 0E_{16}$, represented in (6) as a matrix multiplication (Daemen & Rijmen, 1999), (Daemen & Rijmen, 2002), (Parikh & Narkhede, 2016).

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0E_{16} & 0B_{16} & 0D_{16} & 09_{16} \\ 09_{16} & 0E_{16} & 0B_{16} & 0D_{16} \\ 0D_{16} & 09_{16} & 0E_{16} & 0B_{16} \\ 0B_{16} & 0D_{16} & 09_{16} & 0E_{16} \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad (6)$$

2.1.4. AddRoundKey

In this transformation the state is modified with the bitwise XOR operation with the round key derived from the cipher key and the function Key Schedule. The length of round key is equal to the block length N_b (Daemen & Rijmen, 1999). The inverse of AddRoundKey is called InvAddRoundKey, and is applied in the same way as AddRoundKey applying the keys in reverse order (Rodriguez-Henriquez et al., 2007).

2.2. Key Schedule

Consists in the expansion of the key and in the key selection round (Daemen & Rijmen, 2002). The key expansion specifies how the expanded key is calculated from the cipher key. The number of bits in the expanded key is equal to the block length multiplied by the number of rounds N_r plus one, generating a total of $N_b \times (N_r + 1)$ words, or $N_r + 1$ subkeys, one per each round (Bonadero, Liberatori, Bria & Villagarcía, 2005).

The cipher key is expanded inside of the Expanded key. Round keys are taken from Expanded key as follows: the first round key consists on the initial N_b words, the second on the subsequent N_b words, and so on (Daemen & Rijmen, 1999).

2.2.1 KeyExpansion.

Expanded Key is a four byte linear array denoted by W $[N_b \times (N_r + 1)]$. The first N_k words contain the cipher key, while all other words are defined recursively. KeyExpansion depends of the N_k value and is calculated as in fig. 5, employing the functions subBytes, Rotbyte and Rcon (Daemen & Rijmen, 1999), (Daemen & Rijmen, 2002).

RotByte returns a word that results from a cyclical permutation from the input word, e.g., for an input {a,b,c,d} the output is {b,c,d,a}.

The constant Rcon is independent of N k and is defined in (7) as:

$$Rcon[i]=(RC[i],00_{16},00_{16},00_{16}) \quad (7)$$

where RC[i] represents an element in GF (2^8) with value $x^{(i-1)}$ such that:

$$RC[1]=x^0=01_{16} \quad (8)$$

$$RC[2]=x^1=02_{16} \quad (9)$$

$$RC[j]=x\times RC[j-1]=X^{j-1},\ j>2 \quad (10)$$

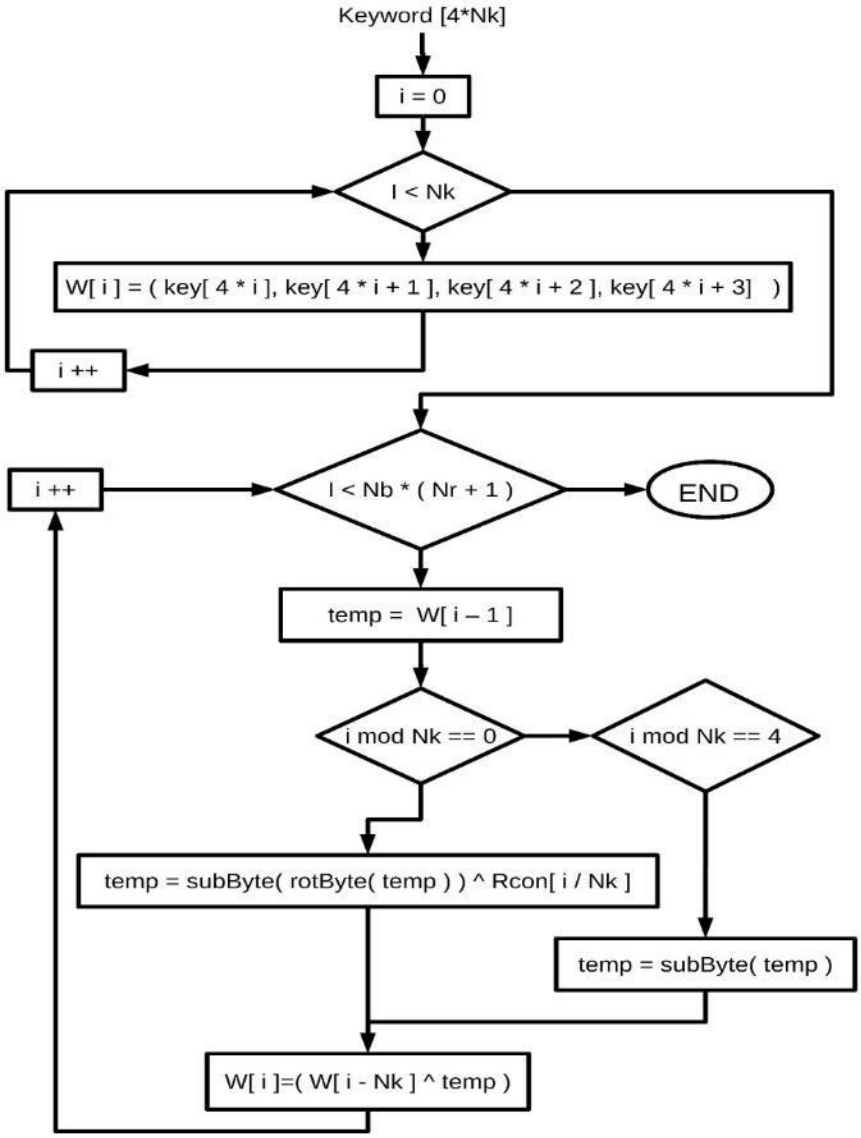

Figure 5. *Flowchart diagram for KeyExpansion function*

3. Proposed Method

The modification consists in substituting the AES S-box for an Artificial Neural Network (ANN) that solves the transformation using as input the corresponding multiplicative inverse value GF (2^8) of the original S-box input value. To obtain the corresponding inverse a lookup table is used. The S-box is substituted for a module formed by a table with the inverse values obtained from (Pelzl & Paar, 2010), (Srebrny, Kościelny & Kurkowski, 2013) and a neural network as is shown in fig. 6. With this method two advantages are obtained, the first one is that the values of the S-box are hidden, and the second one is that it's possible to change the values of the S-box just by a simply changing the weights.

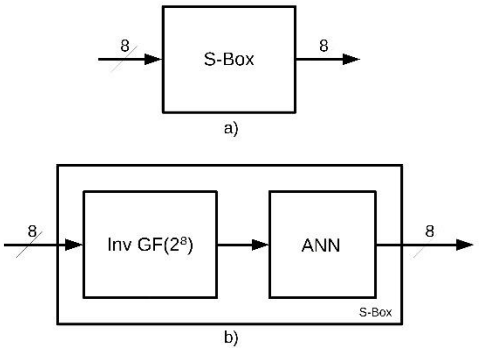

Figure 6. *a) S-box representation. b) Representation of the S-box proposed*

The neural network topology was proposed by means of observation. The transformation is performed bitwise, nevertheless another arrangement is also acceptable. The neural network consists of eight subnetworks, one per bit, each one as illustrated in fig. 7 is composed by seven perceptron neurons in three layers: input layer, hidden layer and output layer. Based on neural networks that perform AND and XOR behaviors each neuron has two inputs and a pulse activation function given by (11).

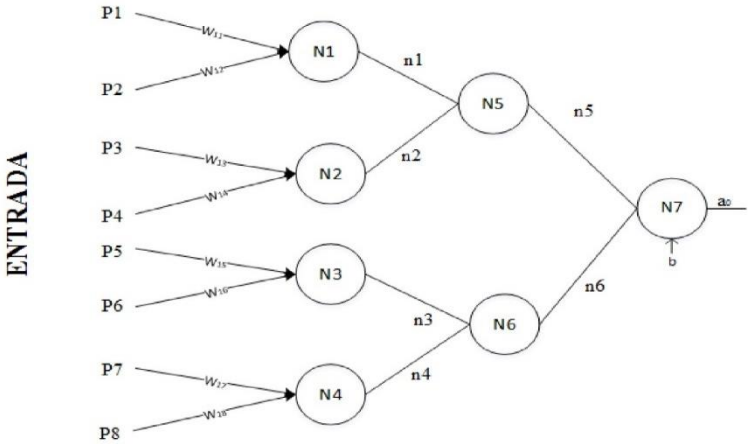

Figure 7. *Neural network with one bit output*

$$f(x)=\begin{cases} if & x=1, \quad 1 \\ if & x\neq 1, \quad 0 \end{cases} \tag{11}$$

The circuit implementation was developed in HSPICE which is an electric circuit simulator (synopsys, 2003), (Piuri, 1991). In hardware implementation, Operational Transconductance Amplifiers (OTA) are used as proposed in (Kawaguchi, Umeno & Ishii, 2014), (Ghosh, LaCour & Jackson, 1994) in order to manage current signals and simplify the sum of the synaptic weights.

The OTA is a voltage controlled current source (VCCS). Its main characteristics are high input impedance and high output impedance (Barclay & Wood, 1994), (Qing-Lin, Jian-You & Mei-Lun, 1991). The OTA macromodel is shown in fig. 8, where V_{in1} and V_{in2} are the voltage inputs, the voltage difference of these sources is reflected in nodes a and b.

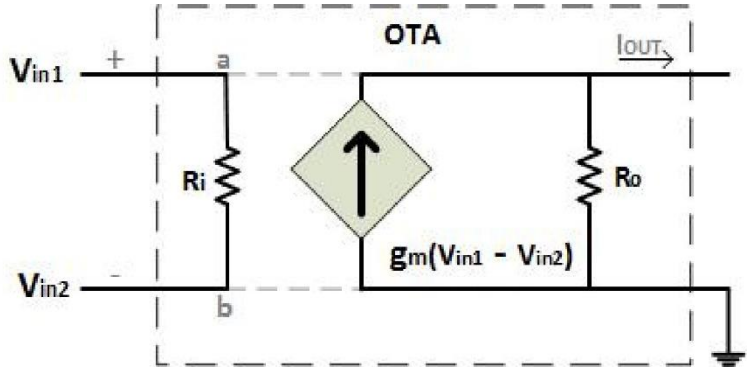

Figure 8. *Macromodel for the Operational Transconductance Amplifier*

The output current I_{out} is proportional to the difference between these voltages as in eqn. 12.

$$I_{out}=g_m\left(V_{positive}-V_{negative}\right) \tag{12}$$

where g_m is the transconductance gain, V_{in1} the positive input voltage, V_{in2} the negative input voltage and I_{out} the output current.

The OTA is used to represent the neuron inputs, converting (in the input layer) or keeping (in the remaining layers) the input signal into a current signal and using the amplifiers gain (g_m) as the corresponding synaptic weight. The signals are summed by simply connecting the OTAs outputs to a wire line which is then the input to the activation function.

4. Simulations

The proposed network was simulated in Matlab, where it was tested and the expected operation for the S-box specified for AES was verified. An implementation using OTAs in HSPICE was performed, where the gain is equivalent to the corresponding weights. Simulating the electric behavior of the system. In the next subsections details of its implementation and results are given.

4.1. Simulation and Results in MATLAB

In the simulation the inverse value in GF(2^8) was used as input of the system and the results were compared and verified with its corresponding S-box values. For a better visualization of the results, the binary values were converted to decimal and are presented in fig. 9 highlighting that the values obtained correspond to those expected with an error of 0%.

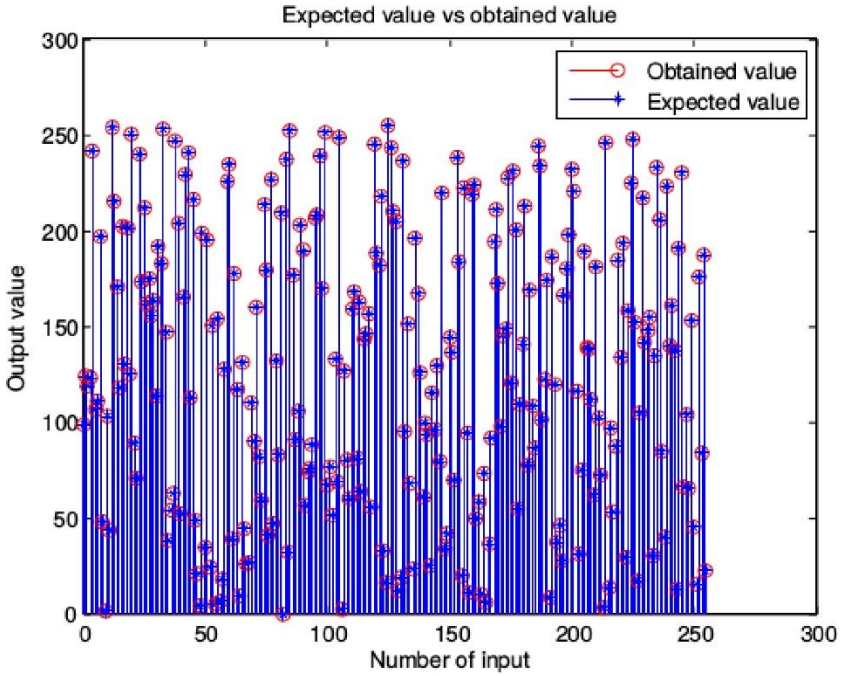

Figure 9. *Expected vs. obtained values. Inputs from 0 to 255*

The synaptic weights used are shown in table 3, these values were obtained from neural networks with AND and XOR behaviors, hence there was no previous training of the network.

Network no.	Synaptic weights	
Bit 0	Input layer	$n_{1,1} = 1, n_{1,2} = 0, n_{2,1} = 0, n_{2,2} = 0, n_{3,1} = 1, n_{3,2} = 1, n_{4,1} = 1, n_{4,2} = 1$
	Hidden layer	$n_{5,1} = 1, n_{5,2} = 1, n_{6,1}=1, n_{6,2}=1$
	Output layer	$n_{7,1} = 1, n_{7,2} = 1$
	Bias	$n_1 = 0, n_2 = 0, n_3 = 0, n_4 = 0, n_5 = 0, n_6 = 0, n_7 = 0$
Bit 1	Input layer	$n_{1,1} = 1, n_{1,2} = 1, n_{2,1}=0, n_{2,2}=0, n_{3,1} = 0, n_{3,2} = 1, n_{4,1}=1, n_{4,2}=1$
	Hidden layer	$n_{5,1} = 1, n_{5,2} = 1, n_{6,1}=1, n_{6,2}=1$
	Output layer	$n_{7,1} = 1, n_{7,2} = 1$
	Bias	$n_1 = 0, n_2 = 0, n_3 = 0, n_4 = 0, n_5 = 0, n_6 = 0, n_7 = 0$
Bit 2	Input layer	$n_{1,1} = 1, n_{1,2} = 1, n_{2,1}=1, n_{2,2}=0, n_{3,1} = 0, n_{3,2} = 0, n_{4,1}=1, n_{4,2}=1$
	Hidden layer	$n_{5,1} = 1, n_{5,2} = 1, n_{6,1}=1, n_{6,2}=1$
	Output layer	$n_{7,1} = 1, n_{7,2} = 1$
	Bias	$n_1 = 0, n_2 = 0, n_3 = 0, n_4 = 0, n_5 = 0, n_6 = 0, n_7 = 0$
Bit 3	Input layer	$n_{1,1} = 1, n_{1,2} = 1, n_{2,1}=1, n_{2,2}=1, n_{3,1} = 0, n_{3,2} = 0, n_{4,1}=0, n_{4,2}=1$
	Hidden layer	$n_{5,1} = 1, n_{5,2} = 1, n_{6,1}=1, n_{6,2}=1$
	Output layer	$n_{7,1} = 1, n_{7,2} = 1$
	Bias	$n_1 = 0, n_2 = 0, n_3 = 0, n_4 = 0, n_5 = 0, n_6 = 0, n_7 = 0$
Bit 4	Input layer	$n_{1,1} = 1, n_{1,2} = 1, n_{2,1}=1, n_{2,2}=1, n_{3,1} = 1, n_{3,2} = 0, n_{4,1}=0, n_{4,2}=0$
	Hidden layer	$n_{5,1} = 1, n_{5,2} = 1, n_{6,1}=1, n_{6,2}=1$
	Output layer	$n_{7,1} = 1, n_{7,2} = 1$
	Bias	$n_1 = 0, n_2 = 0, n_3 = 0, n_4 = 0, n_5 = 0, n_6 = 0, n_7 = 0$
Bit 5	Input layer	$n_{1,1} = 0, n_{1,2} = 1, n_{2,1}=1, n_{2,2}=1, n_{3,1} = 1, n_{3,2} = 1, n_{4,1}=1, n_{4,2}=0$
	Hidden layer	$n_{5,1} = 1, n_{5,2} = 1, n_{6,1}=1, n_{6,2}=1$
	Output layer	$n_{7,1} = 1, n_{7,2} = 1$
	Bias	$n_1 = 0, n_2 = 0, n_3 = 0, n_4 = 0, n_5 = 0, n_6 = 0, n_7 = 0$
Bit 6	Input layer	$n_{1,1} = 0, n_{1,2} = 0, n_{2,1}=1, n_{2,2}=1, n_{3,1} = 1, n_{3,2} = 1, n_{4,1}=1, n_{4,2}=0$
	Hidden layer	$n_{5,1} = 1, n_{5,2} = 1, n_{6,1}=1, n_{6,2}=1$
	Output layer	$n_{7,1} = 1, n_{7,2} = 1$
	Bias	$n_1 = 0, n_2 = 0, n_3 = 0, n_4 = 0, n_5 = 0, n_6 = 0, n_7 = 0$
Bit 7	Input layer	$n_{1,1} = 0, n_{1,2} = 0, n_{2,1}=0, n_{2,2}=1, n_{3,1} = 1, n_{3,2} = 1, n_{4,1}=1, n_{4,2}=1$
	Hidden layer	$n_{5,1} = 1, n_{5,2} = 1, n_{6,1}=1, n_{6,2}=1$
	Output layer	$n_{7,1} = 1, n_{7,2} = 1$
	Bias	$n_1 = 0, n_2 = 0, n_3 = 0, n_4 = 0, n_5 = 0, n_6 = 0, n_7 = 0$

Table 3 *Synaptic Weight Values*

4.2. HSPICE Implementation and Results

According to the structure proposed in fig. 7 the architecture shown in fig. 10 is implemented in HSPICE, where V1 through V8 represent the input signals, the weight, W, are represented by the transconductance of the OTAs, the sums are represented by linking the OTAs outputs, and finally the activation function described in (11) is applied.

The structure in fig. 10 has one bit output, hence it's necessary to replicate the structure in order to have an eight bit output. It should be noted that it is not necessary to replicate the voltage sources and their resistance, i.e. the inputs, only the current source, their resistance and the activation functions.

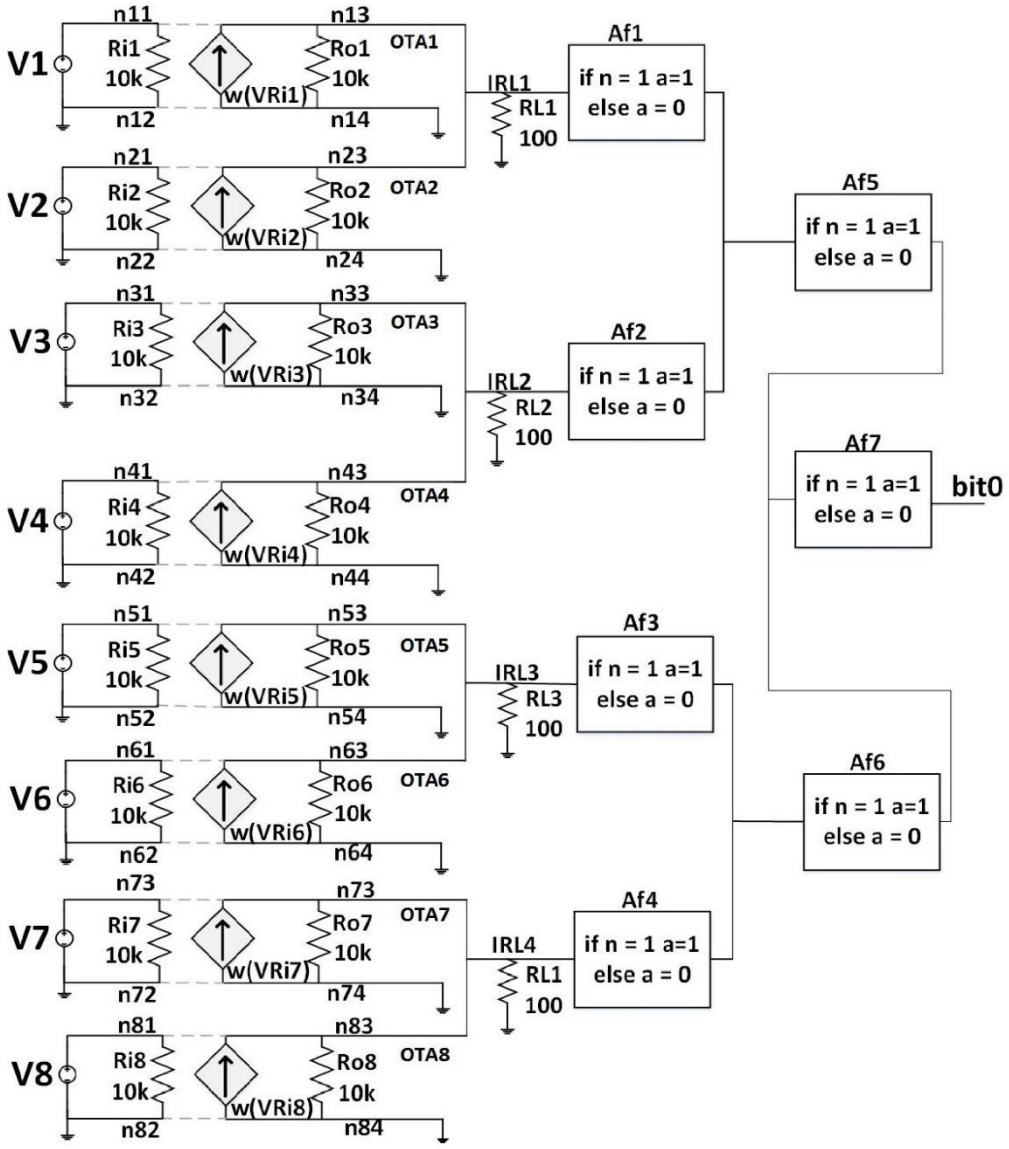

Figure 10. *S-box structure with 1 bit output*

Circuit operation steps

1. The input value is placed in the voltage sources V1 through V8 for the S-box value that wants to be obtained.
2. The voltage difference between nodes n11 and n12 is the voltage in source V1. This difference is multiplied by the gain (weight). This is repeated in voltage source V2 to V8.
3. Since the outputs from the OTAs are given in current, they can be summed by joining them as follows:
   OTA1 output and OTA2 output are linked in Irl1
   OTA3 output and OTA4 output are linked in Irl2
   OTA5 output and OTA6 output are linked in Irl3
   OTA7 output and OTA8 output are linked in Irl4
4. Activation function (11) is applied in Af1 through Af4.
5. Af1 output is linked with Af2, and Af3 with Af4
6. Activation function is applied in Af5 and Af6
7. Af5 and Af6 outputs are linked
8. 8) Activation function is applied in Af7
9. 9) Af7 output corresponds to bit0

As mentioned previously, the structure is replicated to obtain the eight output bits, therefore the same steps are repeated to obtain bit1 to bit7.

To verify the circuit operation, tests were performed with the input values shown in table IV, the table displays some of the values found in the S-box and the result to those inputs, the next two columns show the input value for the proposed network which corresponds to the multiplicative inverse in GF(2^8) and the result obtained from that input. The results obtained from the network are identical, thus the operation of the network is validated.

In figs. 11 and 12 the results obtained from the circuit for four inputs of the table are shown.

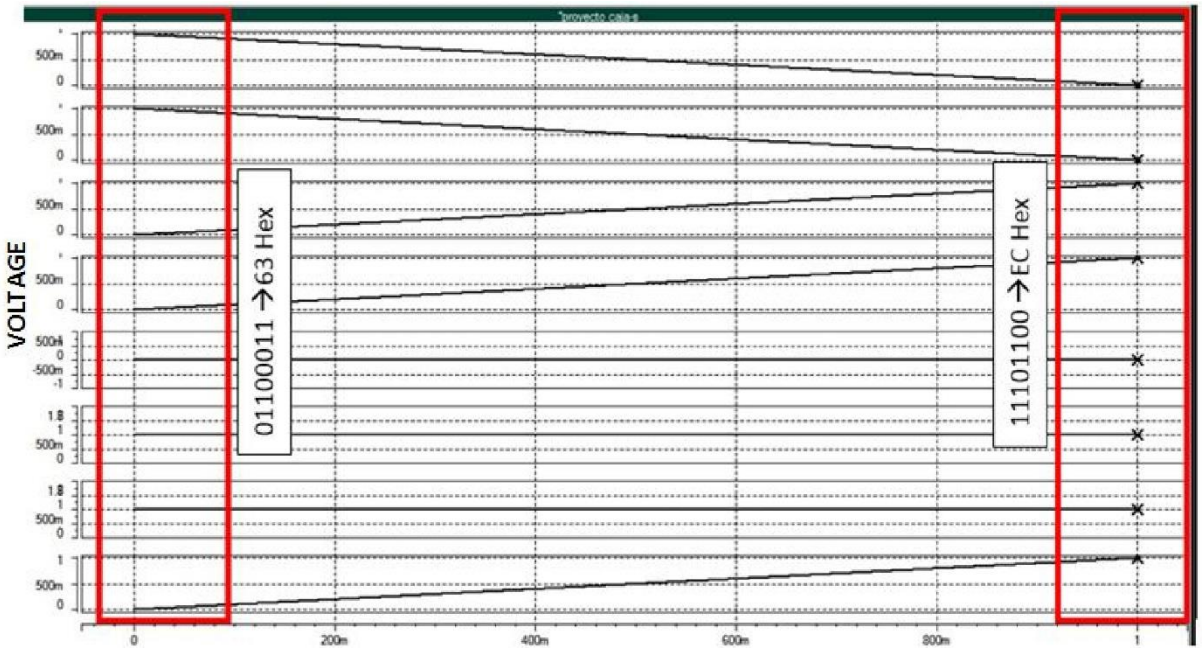

Figure 11. *Obtained result for input 00 16 and 80 16 in GF (2^8)*

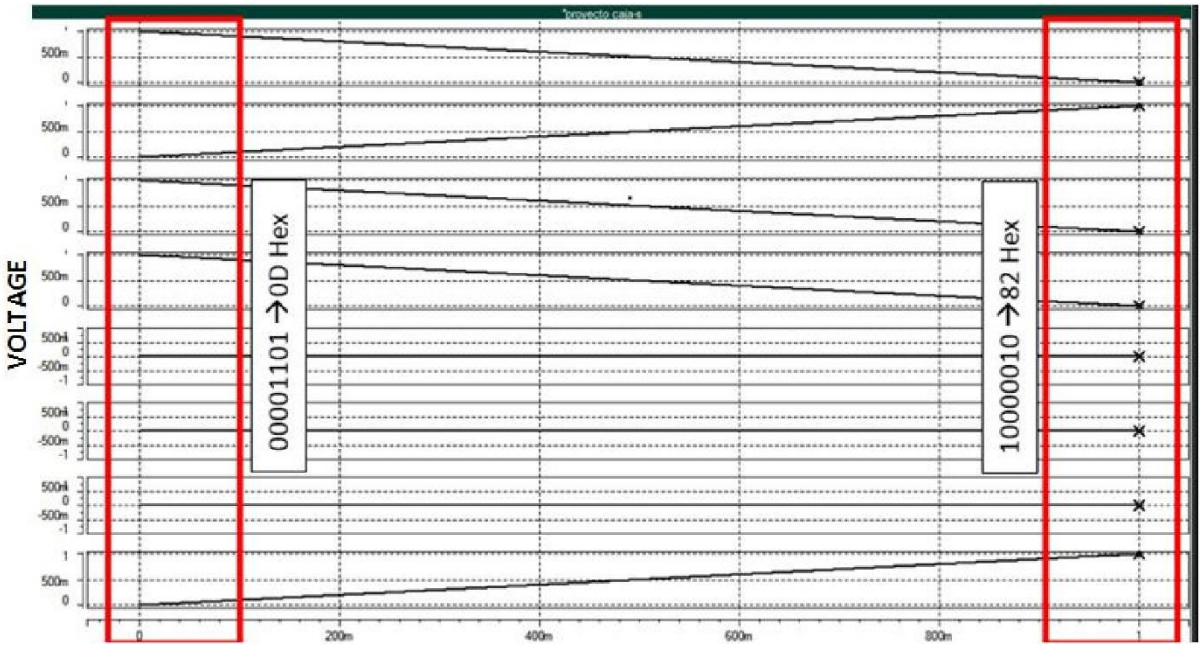

Figure 12. *Obtained result for input 34 16 and 11 16 in GF (2^8)*

e.g. On the left side in fig. 11 the obtained result from the circuit to input 00 16 in GF(2^8) is 63_{16} , the result is verified in table 4. Similarly on the left side the result EC_{16} is obtained for an input 80_{16} in GF(2^8).

S-Box input(Hex)	Result(Hex)	GF(2^8) input(Hex)	Result(Hex)
0	63	0	63
11	82	B4	82
22	93	5A	93
33	C3	6C	C3
44	IB	2D	1B
55	FC	24	FC
66	33	36	33
77	F5	3C	F5
88	C4	9B	C4
83	EC	80	EC
F3	0D	34	0D
C4	1C	DA	1C
5D	4C	EC	4C
E7	94	AD	94
8F	73	A4	73
78	BC	B6	BC
BD	7A	BC	7A
CC	4B	1B	4B

Table 4. *Test Values For The Circuit Implemented In HSPICE*

5. Conclusion

An implementation of an S-box using a neural network in MATLAB and HSPICE is presented, this neural network is based on the operations used to obtain the values of the S-box through 8 perceptron subnetworks and a lookup table with the inverse in GF(2^8). Even if this method of calculating S-box values for AES does not present an advantage reducing resources, since storing the inverse values for each possible input represent hundred percent of the necessary resources to store the original S-box, the values computed by a neural network offers greater security by maintaining the transformation values hidden and using a distractor or an apparently S-box that contains the inverse values in GF(2^8). The simulation results show that the implementation presents a null error, thereafter if the neural network were applied, it will not show changes in the results expected within the encryption algorithm because it simulates without error the operation of the S-box.

References

Barclay M. & Wood J., (1994) A SPICE macromodel for operational transconductance amplifiers. *IEE Colloquium on Analogue Signal Processing*, London, 1994, pp. 1/1-1/4.

Bonadero J., Liberatori M., Bria O. & Villagarcía-Wanza H. (2005) Expanción de la clave en rijndael: diseño y optimización en vhdl. In XI Workshop IBERSHIP.

Daemen J. & Rijmen V. (1999) AES proposal: Rijndael.

Daemen J. & Rijmen V. (2002) The design of Rijndael: AES - the Advanced Encryption Standard. Springer-Verlag.

Ghosh J., LaCour P. & Jackson S. (1994) Ota based neural network architectures with on-chip tuning of synapses. In Proceedings of 7th International Conference on VLSI Design, pages 71–76.

Katz J. & Lindell Y. (2008) Introduction to Modern Cryptography. Chapman & Hall/CRC cryptography and Network Security.

Kawaguchi M., Umeno M. & Ishii N. (2014) The two-stage analog neural network model and hardware implementation. In 2014 IIAI 3rd International Conference on Advanced Applied Informatics, pages 936–941.

Noughabi M. N. A. & Sadeghiyan B. (2010) Design of s-boxes based on neural networks. In 2010 International Conference on Electronics and Information Engineering, volume 2, pages V2–172–V2–178.

Oukili S., Bri S., & Kumar A. V. S. (2016) High speed efficient fpga implementation of pipelined aes s-box. In 2016 4th IEEE International Colloquium on Information Science and Technology (CiSt), pages 901–905.

Parikh P. & Narkhede S. (2016) High performance implementation of mixing of column and inv mixing of column for aes on fpga. In 2016 International Conference on Computation of Power, Energy Information and Commuincation (ICCPEIC), pages 174–179.

Pelzl J. & Paar C.. (2010) Understanding Cryptography - A Textbook for Students and Practitioners. Springer-Verlag Berlin Heidelberg, 1 edition.

Piuri V. (1991) The use of the electrical simulator spice for behavioral simulation of artificial neural networks. In 1991 Proceedings of the 24th Annual Simulation

Symposium, pages 18–29.

Qing-Lin Sun, Jian-You Liu & Mei-Lun Liu, (1991) An improved nonlinear macromodel of OTA, *1991 International Conference on Circuits and Systems*, Shenzhen, *China.* pp. 906-908 vol.2.

Rîncu C. & Iana V. (2014) S-box design based on chaotic maps combination. In 2014 10th International Conference on Communications (COMM), pages 1–4.

Rodriguez-Henriquez F., Saqib N.A., Díaz A., & Koc CK. (2007) Cryptographic Algorithms on Reconfigurable Hardware. US: Springer.

Srebrny M., Kościelny C. & Kurkowski M. (2013) Modern Cryptography Primer, Theoretical Foundations and Practical Applications. Springer-Verlag Berlin Heidelberg, 1 edition.

synopsys. (2003) HSPICE Simulation and Analysis User Guide.

Thomson K, Siva. N, & Priya S. (2014) Implementation of low-area s-box based on normal basis. In 2014 International Conference on Electronics and Communication Systems (ICECS), pages 1–4.

Yang M., Wang Z., Meng Q., & Han L. (2011) Evolutionary design of s-box with cryptographic properties. In 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications Workshops, pages 12–15.

Zhang X., Chen F., Chen B., & Cao Z. (2015) A new scheme for implementing s-box based on neural network. In 2015 International Conference on Computational Science and Computational Intelligence (CSCI), pages 571–576.

Notas biográficas:

Jaime David Rios Arrañaga received the B. degree in Eng. in communications and electronics in 2014, currently pursuing a M.Sc. degree in electronics and computer science engineering at the University of Guadalajara. His current research is on cryptographic systems in reconfigurable hardware.

Janneth Alejandra Salamanca Chavarin received the B. degree in Eng. In communications and electronics in 2014, currently pursuing a M.Sc. degree in electronics and computer science engineering at the University of Guadalajara. Her current research interest is biological neural networks.

Juan José Raygoza Panduro Ph.D. in Computer Science and Telecommunications from Autonomous University of Madrid, Spain. He specializes in the design of digital architecture based on FPGAs, Microprocessors, VLSI, embedded system and bioelectronics, Neuroengineering. Research Professor at University of Guadalajara.

Edwin C. Becerra Alvarez Ph.D. degree in Microelectronics from University of Seville, Spain. His current research interests are on integrated CMOS design, transceiver design and embedded systems. Research Professor at CUCEI; University of Guadalajara