An Optimized Implementation of the S-Box using Residues of Prime Numbers

Eltayeb Salih Abuelyman & Abdul-Aziz Sultan Alsehibani
CCIS, Prince Sultan University, Riyadh 11586, Saudi Arabia

Summary

The S-Box is a major step of the Advanced Encryption Standard (AES). In the current AES, the step uses a table lookup based on the Galois Field GF(2^n) to map the plaintext to the ciphertext with confusion. This paper proposes an AES S-Box implementation that is based on modified lookup table. The proposed platform for implementing the S-Box builds upon the hypothesis that the set of residues of any prime number forms a mathematical field. Two sparse lookup tables with irregularly distributed entries are used to add more confusion to the process. The inverse of any entry in either of the tables is stored on one of them. The goal of this paper is therefore twofold: the first is to find an alternative to the Galois Field implementation and the second is to add more confusion to the S-Box implementation.

Keywords: Modulo Arithmetic, Mathematical Field, VHDL, Hardware Implementation, S-Box Confusion and Diffusion.

1. Introduction

The need for encryption continues to grow directly proportional to the amount of data transmitted in plaintext. The situation is worsened by the incredible amount of time spent in hacking and the sophistication of tools used in the process. Another point of concern is that software-based implementations of cryptographic algorithms fall short of achieving the expected performance for lower ranges of transmission speeds. The significance and applicability of hardware-based implementations of cryptographic algorithms is therefore of interest to researchers including members of the VHDL design community [1]. First, a brief introduction to the AES is provided.

1.1 The Advanced Encryption Standard

In Reference [2] the authors stated that "The Advanced Encryption Standard (AES) committee solicited proposals for an encryption algorithm that would become the first choice for most situations requiring a block cipher". Consequently, several algorithms were submitted and Rijndael was chosen by the American National Institute of Standards and Technology (NIST)[3].

Manuscript received April 5, 2008
Manuscript revised April 20, 2008

1.2 The Rijndael Algorithm

For Rijndael, the length of both the block to be encrypted and the encryption key are not fixed. They can be independently specified to 128, 192 or 256 bits. The number of rounds, however, varies according to the key length. It can be equal to 10, 12 and 14 when the key length is 128 bits, 192 bits and 256 bits, respectively [4]. The basic components of Rijndael are simple mathematical, logical, and table lookup operations. The latter is actually a composite function of an inversion over Galois Field (GF) with an affine mapping. Such structure makes Rijndael suitable for hardware implementation [2]. Nevertheless, both hardware and software implementations have their own drawbacks. Hardware implementation is rigid as a block and key sizes must be held at fixed values. However, the running time is better compared to its software counterpart. All in all, Rijndael is considered to be the fastest algorithm in terms of the critical path between plaintext and cipher-text [2].

In general, most of the low-level components of block ciphers emphasize the functional aspect of the likes of the S-Box. According to Vincent Rijmen, the co-designer of the Rijndael algorithm, the Advanced Encryption Standard (AES), which is based on the S-BOX is still superior in security [1]. In fact, their algorithm enjoys extremely high level of confusion and diffusion. Furthermore, the resistance of the inverse function in the Galois Field GF(2^n) [20] to linear, differential and higher-order differential attacks is exceptional according to Rijmen. However, one of the disadvantages of AES is the simplicity of description in GF (2^n), which is also the field in which the diffusion layer is linear. The AES designers believe that this may create uneasy feelings, but they are not aware of any vulnerability thereof. Should such a vulnerability exist, they suggest the replacement of the GF(2^n) by another field that has similar properties, but is not algebraic over GF(2^n).

A potential vulnerability that will be dealt with here is not directly related to the aforementioned discussion. The vulnerability discussed here stems from storing the S-Box in an attempt to avoid the overhead of tedious real time computations. The authors of this paper demonstrated in a previous paper that arithmetic modulo prime numbers provides a valid, less complex alternative to real time computation of S-Box inverses [18]. One may argue that a platform with less computation complexity compared to the Galois Field option may suffer from increase in vulnerability. However, such tradeoff is acceptable considering the gain in the amount of information stored and the consequences thereto. However, more work is needed to confirm that the vulnerability of the resulting platform is less relative to that of the original implementation via Galois Fields.

This paper proposes the design of residues of a prime number based S-Box. The rest of the paper is organized as follows. In section 2, a literature survey is presented; in section 3, the S-Box implementation using residues of a prime number is introduced. The conclusion is given in section 4.

2. Literature Survey

Ichikawa, Kasuya, and Mastui published a paper that evaluates hardware implementations of the AES finalists: Twofish [13]; Serpent [14]; RC6 [15]; Mars [16]; and Rijndael [17]. Commenting on Mars, the authors stated two problems: the keyed transformations take a long time and, the algorithm is very complex. They also concluded that RC6 gives a poor performance since the critical path is long. The RC6, according to them, did not satisfy the need for fast encryption. They believe Serpent has the best security but it requires the largest circuit. They also believed that Twofish has quite a long critical path. In their paper, Pawel Chodowiec and Kris Gai gave data supporting Rijndael [6]. The throughput of Rijndael came second. However, considering all the other criteria, Rijndael was found to be the best. Ian Harvey discussed the selection of encryption algorithm in practical situations in his paper [7]. AES finalists are compared based on the factors considered for algorithm selection. Bryan Weeks et al presented an overview of the methods and architectures used for the AES hardware comparison in their paper [4]. In general, throughput, area and latency are the characteristics considered for design tradeoffs in hardware engineering. The five finalists were examined from the standpoint of minimum area and maximum throughput. Interested readers may consult reference [4] for further details. A. Satoh, S. et al presented an AES hardware implementation they considered to be efficient in their paper [8]. However, the main drawback of their architecture is the critical path time. The SubBytes, MixColumns and AddRoundKey transformations are done for one column within one clock cycle. This increases the critical path time. In the next subsection, a survey of some of the VHDL implementations is presented.

2.1 VHDL implementations

Algotronix AES Core [9] represents the second generation of their AES VHDL technology. It is a stable implementation of the entire algorithm. It offers competitive density and performance on all the main Field Programmable Gate Arrays (FPGA) families from Xilinx, Altera and Actel. It is supplied as synthesizable source code to allow for customer code review in security sensitive applications. The core is highly configurable with many implementation options but unlike most competitive products, this is achieved using VHDL generic parameters and does not require customizing the VHDL code.

In their paper, Arda Yurdakul et al [10] discussed the design and implementation of three configurable and flexible cores of Rijndael. The three cores are; an encryptor, a decryptor and a combined encryptor-decryptor. These cores support not only the AES, but also the whole Rijndael algorithm. Another feature of the cores is that they are all designed using Electronic Code Book (ECB) mode, meaning that every single data block is encrypted and decrypted independently from each other. Since ECB is the basic element of all other main modes such as Cipher Block Chaining (CBC), Cipher Feedback (CFB) and Output Feedback (OFB), it is easy to extend their design and implement the other modes. All the modules in these flexible cores are realized using VHDL language. Some modules are designed by using behavioral style and some are designed using Register Transfer Language. In the next section the implementation of the modulo arithmetic based AES is presented.

2.2 Concerns about the Rijndael

Generally speaking for the hardware implementations of Rijndael, Ian Harvey [2] states that the average time for one lookup table is 3.2 nanoseconds for Rijndael (8x8). If one is able to optimize the S-Box lookup process, then the speed can be greatly increased. The S-Box computation is the most time consuming operation. This is the case because it is required in every round. Current implementations pre-compute the S-Box and store it on a Read Only Memory (ROM). However, in a highly sensitive data environment, storing such information poses a threat to its security. To harden against such vulnerability, at least a part of the S-Box entries should be dealt with differently.

Another concern is the speed of computation. To speed up real-time S-Box construction, an environment other than the Galois Field (GF) must be used. The residue arithmetic environment takes significantly less time and space compared to the GF as argued in reference [11]. Legitimate candidates for the residues arithmetic are numbers that are residues of powers of two, and numbers that are residues of a prime. The former candidate results in table entries that are symmetric and produces columns that follow systematic patterns. In reference [19] authors took advantage of these patterns and reduced the table to only a single row. That was an interesting move; nonetheless, for devices with limited computational resources storing the S-Box may become a better option. This paper proposes a compromise which stores carefully selected parts of the S-Box entries. The choice of the stored entry enables the deduction of the rest. The process uses arithmetic on residues of a prime number to generate entries of the S-Box as discussed in the next section.

3. S-Box generation using residue of a prime arithmetic

The proposed implementation is based on residues of a prime number. Table 1 below shows the complete S-Box with 256 entries. These entries are the residues of the prime number 257. The choice of 257 is logical because residues from 1 to 255 have unique inverses. Furthermore, these residues can be used for all block sizes of the AES; that is, they can be used for the 256, 192 and 128 bits blocks. The residues handling within the S-Box is discussed next.

3.1 The Residues of 257

The row and column headers of table 1 are hexadecimal digits. For short, an S-Box table lookup is hereafter referred to by S. Without loss of generality, and given the hexadecimal digits i, j, k, and l, it can be seen that if S(i, j) = kl, then S(k, l) = ij. In this case S(i, j) defines a look up for the inverse of the two digits hexadecimal number ij. Similarly, S(k, l) is a lookup for the two digits inverse of the hexadecimal number kl. Both ij and kl are stored on the table. Without loss of generality, let us take ij = A7. The inverse of ij denoted by kl is equal to ED. This can easily be verified by checking the entry in i^{th} row and j^{th} column. One can also verify that the reverse is also true. That is, the inverse of ED is A7. Once again, both numbers are stored on the table.

To address the vulnerability concern of storing the S-Box table, one needs to store only some of the entries and figure out a way to determine the rest. Fortunately, a 50% reduction of table 1 is achievable due to the fact that all the double digits hexadecimal numbers and their inverses coexist on the same table. Therefore, the best possible reduction is to store only half of the numbers and their inverses and omit the other half. Obviously, such reduction will result in a miss ratio that equals the reduction percentage.

When the lookup of the reduced table fails, a deduction process must be used to determine the sought value. In the next section we will discuss reducing the table.

3.2 Reducing the lookup table

Without loss of generality, let us consider a couple of lookup operations on table 1. It is clear that S(F, A) = 6E and S(6, E) = FA. This implies that both numbers are inverses of each other and both are stored on the table. Since every double digits hexadecimal number and its inverse are stored, it is logical to reduce the table by eliminating half of the numbers and their inverses. Table 2 below shows the reduced version of table 1. While the eliminated half is unknown, one cannot determine whether or not a number is stored on the reduced table.

The reduced table contains 128 entries or 50% of table 1, hence, not every lookup operation will be successful. Obviously in the case of a lookup hit, only one operation is sufficient. However, in the case of a miss, two operations will be necessary. The following two steps can be used to lookup the reduced table.

1. If S(F, A) returns a number then that number is the inverse of FA and the second step is not required
2. a) Search the reduced table for the cell containing FA and return the row and column headers of that cell.

   b) Use the returned row and column headers as the most and least significant digits respectively of the inverse of FA

For example, performing S(F, A) on the reduced table will return a blank (a miss) indicating that the inverse of FA is not stored on the table but FA is. However, the cell in which FA is stored in unknown and a search for it is necessary. When the cell in which FA is stored is found, a number is constructed from the row and column headers of the cell. The row header is the most significant digit and the column header is the least digit. The constructed number is the inverse of FA.

Applying the algorithm for the number FA will result in the number (6E) as its inverse. While step 2(b) is acceptable, step 2(a) is time consuming. Particularly, the average search to locate a value in the 256 possible cells is 128 tries which may not be acceptable for devices with low computational resources. Handling this concern is discussed next.

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

Table 1 Residues of the prime number 257

0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	*0*	01	*81*	56	*C1*	67	*2B*	93	*E1*	C8	*B4*	BB	*96*	B2	*CA*	78
1	*F1*	79	*64*	E6	*5A*	31	*DE*	BE	*4B*	48	*59*	EE	*65*	C3	*3C*	C7
2	*F9*	94	*BD*	EB	*32*	84	*73*	91	*2D*	A3	*99*		*6F*		*5F*	AF
3	*A6*			7E	*AD*	61	*77*	F3	*B3*	F8	*E2*	3D			*E4*	66
4	*FD*	57	*4A*	EA	*DF*	95	*F6*	B5		A9			*BA*	F7	*C9*	F4
5	*97*	A5	*D2*	60	*CD*	7F			*B8*			D1	*B0*	98	*D8*	
6				87					*D7*	A4	*B1*	F5	*BC*	E0	*FA*	
7	*DA*	74	*7C*			86	*9F*				*9E*	8C		DC		
8	*FF*		*AC*	CE		8F			*F0*	F2	*CB*	62		90	*DB*	
9			*D5*								*FC*	C2	*E5*	EF		
A	*CC*	AE	*D3*					ED	*E7*		*C0*	FE				
B							*E9*	BF		E8						
C					*C6*	E3										DD
D	*EC*				*D9*		*FB*									
E																
F																

Table 2 Reduced version of the residues of 257

3.3 Generating an inverse for the reduced table

A second table complementing the reduced table can be generated to avoid searching the whole reduced table at the aforementioned step 2(b) upon a lookup miss. The second table is called the complement of the reduced table and is manipulated using the following rules:

I) If S(i, j) is a miss, then the hexadecimal number ij is found on the reduced table and its inverse can be looked up in the complement table.

II) The parameters for looking up the complement table for the inverse of the hexadecimal number ij in rule I) would be the 1's complement of the binary representations of the number.

The second rule uses the complement of each of the parameters in a miss case when the reduced table is looked up. For example, in section 3.1 and using table 1, it has been shown that S(F, A) = 6E and S(6, E) = FA. Now, using table 2 one can verify that S(F, A) is a miss and S(6, E) is a hit. It is clear that one has to use rule number 2 in locating the inverse of FA. A quick glance at table 3 shows that the inverse of FA is found looking up S(0,5). Noticeably, 0 and F are the 1's complements of F and A respectively.

0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	80	*AB*	40	9A	*D6*	6E	*20*	39	*4D*	46	*6B*	4F	*37*	89	*10*	88
1	*9D*	1B	*A7*	D0	*23*	43	*B6*	B9	*A8*	13	*9C*	3E	*C5*	3A	*08*	6D
2	*44*	16	*CF*	7D	*8E*	70	*D4*	5E	*68*		*92*		*A2*	52	*5B*	
3		83	*54*	A0	*8A*	0E	*4E*	09	*1F*	C4			*1D*	9B	*04*	AA
4	*B7*	17	*22*	6C	*0B*	4C		58			*47*	0A	*38*	0D	*6A*	5C
5	*2F*	A1	*34*	82			*49*			30	*51*	69	*29*			
6	*76*	7A					*2A*	5D	*50*	0C	*45*	21	*07*		*27*	8D
7	*85*			7B					*63*	75		25			*02*	
8	*55*	33		72			*11*	0F	*36*			71	*26*			
9	*2C*								*05*	3F	*1C*	12		8B	*35*	53
A	*2E*					14	*1A*		*41*	03						
B					*18*	42		19								
C			*3B*	1E										24	*15*	
D			*28*		*06*											
E																
F																

Table 3. Inverse of the reduced table

4. Conclusion

This paper introduces the concept of a reduced S-Box table and its complement. The dual table solution introduced further confusion to the S-Box process. The proposed solution avoids directly storing the S-Box entries to improve security while providing easy mechanism for finding inverses. On the average, the use of the dual table solution returns the inverse of a number in 1.5 tries. Considering the gain in security, the proposed solution is superior to the option of directly storing all the entries despite the fact that every inverse is found in the first try for the latter case. The proposed solution is also superior to the reduced table version which stores only half of the entries but needs 128 tries on the average to locate an inverse.

References

[1] Swankoski, E.J., Brooks, R.R., Narayanan, V. , Kandemir, M., Irwin, M.J. (2004) “A Parallel architecture for Secure FPGA Symmetric Encryption”

[2] Harvey, I., (2005) “The Effects of Multiple Algorithms in the Advanced Encryption Standard”, nCipher Corporation Ltd., 4’Th January 2000 Retrieved on November 6, 2005

[3] Daemen, J. and Rijmen, V. (2005) “AES Proposal: Rijndael,” Document vers on 2, Date: 03/09/99. Retrieved on October 20, 2005

[4] Bryan Weeks, Mark Bean, Tom Rozylowicz, Chris Ficke, “Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms”, National Security Agency. Retrieved on November 8, 2005

[5] Tetsuya Ichikawa, Tomomi Kasuya, Mitsuru Matsui, “Hardware Evaluation of AES Finalists”, Kamakura Office, Mitsubishi Electric Engineering Company Limited.
Retrieved on October 30, 2005

[6] Pawel Chodowiec and Kris Gaj , “Comparison of the Hardware Performance of AES candidates using reconfigurable hardware” ,spring 2002

[7] Advanced Encryption Standard Development Effort. http://www.nist.gov/aes.

[8] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, “A Compact Rijndael Hardware Architecture with S-Box Optimization,” Proc.Advances in Cryptology—ASIACRYPT 2001, pp. 239-254, 2001.

[9] http://www.algotronix.com/engineering/aes1.html

[10] http://www.cmpe.boun.edu.tr/~yurdakul/papers/OzpinarDSD03.pdf

[11] Abuelyaman, E. “Alternative S-Box Computation Method for AES Environments.” Technical Report, School of Information Technology, Illinois State University, Normal, IL. December 2005

[12] Guy, R. K. "Euler's Totient Function," "Does $\phi(n)$ Properly Divide $n-1$," "Solutions of $\phi(m)=\sigma(n)$," "Carmichael's Conjecture," "Gaps Between Totatives," "Iterations of ϕ and σ," "Behavior of $\phi(\sigma(n))$ and $\sigma(\phi(n))$." §B36-B42 in Unsolved Problems in Number Theory, 2nd ed. New York: Springer-Verlag, pp. 90-99, 1994.

[13] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, “Twofish: A 128-Bit Block Cipher”, 15’Th June, 1998

[14] Ross Anderson, Eli Biham and Lars Knudsen ,” The Case for Serpent” , 24th March 2000

[15] Ronald L. Rivest1, M.J.B. Robshaw2, R. Sidney2, and Y.L. Yin2, “The RC6 Block Cipher”, August 20, 1998

[16] IBM MARS Team, “MARS and the AES Selection Criteria”, May 15, 2000

[17] J. Daemen and V. Rijmen, “AES Proposal: Rijndael ” Document version 2 1999 – May

[18] Abuelyaman Eltayeb and El-Affendi Mohamed “A Real Time S-Box Construction Using Arithmetic Modulo Prime Numbers” Journal of Digital Information Management, Vol 5, No. 6, pp 354-360, December 2007

[19] Abuelyaman Eltayeb and El-Affendi Mohamed “An Optimized Real Time Generation of the S-Box Inverses Using Arithmetic Modulo Powers of Two” International Journal of Computer Science and Network Security, Vol. 7, No 12, pp 240-246, December 2007

Eltayeb Salih Abuelyaman received a PhD degree in Computer Engineering from the University of Arizona in 1988. He served as faculty member at various universities in the US for 18 years before moving to Prince Sultan University in Saudi Arabia where he served as a Faculty Member, the Director of the Information Technology and Computing Services and currently serves as the Dean of the College of Computer and Information Sciences. His current research Interest is in the areas of Computer Networks, Information Security and Database.

Dr. Abdul-Aziz Sultan Alsehibani received a BS and MS degrees in Computer Engineering from King Saud University in 1989 and Syracuse University in 1993 respectively. He received a PhD degree in Computer Science from Syracuse University in 1999. He is currently serving as the Dean of Admissions and Registration. He also served as interim Dean for the College of Computer and Information Sciences form 2006 to 2007. Dr. Alsehibani's research is in the areas of Multimedia and Object Allocations, Computer Architecture, and Networking Security.