A Real Time S-Box Construction Using Arithmetic Modulo Prime Numbers

JDIM
Journal of Digital Information Management

Eltayeb Salih Abuelyman
CCIS, Prince Sultan University, Riyadh 11586, Saudi Arabia
abuelyaman@cis.psu.edu.sa

Mohammed Ahmed El-Affendi
CCIS, Prince Sultan University, Riyadh 11586, Saudi Arabia
affendi@cis.psu.edu.sa

ABSTRACT: *This paper proposes an implementation of the inverse function of the Advanced Encryption Standard using the field of prime numbers instead of the Galois Field originally proposed by Rijndael. The paper will show that the former approach is simpler and requires less execution time and implementation circuitry compared to the latter. The authors analyzed several implementations of the inverse function for the S-Box using various approaches in search for an optimal one. In particular, simulation was used to analyze performances of algorithms for computing the inverse function based on: the arithmetic modulo a power-of-two; arithmetic modulo a power-of-two plus one; and arithmetic modulo a prime number. The simulation revealed that the modulo a prime number approach has the best performance. Furthermore, the analysis revealed that using this approach may enhance security relative to the original approach. The proposed implementation will provide a better alternative that can be embedded in many systems.*

Categories and Subject Descriptors
E.3 [Data Encryption]; F.2 [Analysis of Algorithms and Problem Complexity] G.1 [Numerical Analysis]; Computer arithmetic

General Terms
Encryption, Prime number, Cryptographic algorithms

Keywords: Modulo Prime Arithmetic, VHDL, Hardware Implementation, Diffusion, Pipelining, Parallel Execution

Received 12 Dec. 2006; Revised and accepted 17 Feb. 2007

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 time spent in hacking and the sophistication of the tools used in the process. On the other hand, 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]. In the next subsections, a brief introduction to the AES committee's criteria that lead to the Rijndael algorithm will be reviewed.

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].

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 the block and key sizes must be held at fix 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 critical path between plaintext and cipher-text [2]. This paper proposes the design of a modulo prime-number based AES algorithm. The design will be simulated in a VHDL environment to confirm its superiority. The VHDL modules will not be included in this paper. The rest of the paper is organized as follows: In section 2, a literature survey is presented; in section 3, three modulo arithmetic based techniques are analyzed; the conclusion is given in section 4.

2. Literature Survey

Ichikawa, Kasuya, and Mastui evaluated in "Hardware Evaluation of AES finalists" [5]. The paper 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 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 titled "Comparison of the Hardware Performance of AES candidates using reconfigurable hardware" 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 titled "The Effects of Multiple Algorithms in the Advanced Encryption Standard" [7]. AES finalists are compared based on the factors considered for algorithm selection. Bryan Weeks etal presented an overview of the methods and architectures used for the AES hardware comparison in their paper titled "Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms" [4]. In general, throughput, area and latency are the characteristic 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. etal presented an AES hardware implementation they considered to be efficient in their paper "A Compact Rijndael Hardware architecture with S-Box Optimization" [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 "Configurable Design and Implementation of the Rijndael Algorithm-AES", Arda Yurdakul etal [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.

3. AES Implementation using modulo arithmetic

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 of Rijndael can be greatly increased. The S-Box computation is the most time-consuming operation in Rijndael. Such 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, there is a chance that in a highly sensitive data environment, storing such information may pose a threat to its security. To circumvent such vulnerability, the S-Box values must be computed on a real-time basis. However, using the Galois Fields (GF) renders this option undesirable. To speed up real-time S-Box construction, an environment other than the GF must be used. The reason real time computation of the S-Box is advantageous is two fold: first, when the lookup table is stored for future reference, it is vulnerable to attacks; hence, it is a security concern. Second, if a device doesn't have enough resources, real-time computation of inverses of numbers for the S-Box in Galois Field environment becomes a bottleneck. To overcome these limitations, real time computations of these inverses can be performed using arithmetic modulo some number. The proof that the modulo arithmetic approach is efficient and takes significantly less time and space compared to the GF can be found in reference [11]. Legitimate candidates for the modulo arithmetic are numbers that are powers of two and prime numbers. Henceforth, arithmetic modulo a power of two will be referred to as AM(2^n), and arithmetic modulo a prime will be referred to as AM(P). In the next three subsections we will present analysis of AM(2^n), analysis of AM(2^n+1), and analysis of AM(P).

3.1 Computing inverses modulo (2^n)

The program in appendix 'A' is a Java Script version of one of the VHDL modules for computing the S-Box values. It is intended to give interested readers an idea about the complexity of one member of a suite of modules for the complete process. Without loss of generality, the program was used to compute the inverses of all numbers that are less than 2^4 using modulo 2^4. For convenience, the output is rearranged in table 1 below. The table shows an AM(2^4) mapping of numbers less than 2^4.

Number	Inverse Modulo 16
1	1
2	-
3	11
4	-
5	13
6	-
7	7
8	-
9	9
10	-
11	3
12	-
13	3
14	-
15	15

Table 1. Inverses Modulo 16 mapping the first 15 integers

Only odd numbers on the first row have inverses modulo 16. The inverses are shown in the second row, which also shows a '-' for numbers without inverses. In this case all the even numbers do not have inverses, as expected. The following is a generalization of the relationship between numbers and their multiplicative inverses modulo a power of 2.

Lemma 1

Given any integer n and any number 'a' that is less than 2^n, if 'a' has a multiplicative inverse modulo 2^n , then both 'a' and its multiplicative inverse must be odd numbers.

Proof

If 'a' and 'b' are multiplicative inverses of each other modulo 2^n, then for some integer z less than 2^n the following equation holds:

$$a * b = (z * 2^n) + 1 \quad (1)$$

Since the right hand side is an odd number, it follows that 'a' must be an odd number and 'b' must also be an odd number. This proves that there are no multiplicative inverses for even numbers modulo 2^n. The Eueler's Totiet [12] will be used to prove that every odd number has a multiplicative inverse modulo 2^n. In general, mathematicians use Euler's Totiet function 'Φ' to compute the number of integers that are relatively prime (or multiplicative inverses) to a particular integer n. The function, denoted $\Phi(n)$, is given by the following product:

$$\Phi (n) =$$

$$[(p_1 – 1)^* p_1^{k1-1}] * [(p_2 – 1) * p_2^{k2-1}] * \dots * [(p_m – 1)^*p_m^{km-1}] \quad (2)$$

Where n in this case is expressed in term of its prime factors $[p_1, p_2, \dots p_k]$.

A special case of the Euler's function can be used to find the number of integers that are relatively prime to 2^n. Since a power of 2 has the number 2 as its only prime factor then replacing "n" with "2^n"; "p_1"with the number 2; "k1" with "n" we reach equation 3 .

$$\Phi (2^n) = (2-1)(2^{n-1}) = 2^{n-1} \quad (3)$$

The formula shows that half of the numbers less than 2^n are relatively prime to it. Since no even number is relatively prime to 2^n and there are exactly 2^{n-1} odd number less that 2^n , it follows that all the odd numbers less than 2^n have multiplicative inverses. This completes the proof.

Lemma 2

For any integer n, if we divide the sequence of odd numbers from 1 to "2^{n-1}" into two disjoint subsets where the first contains the sequence of the all odd numbers that are less than 2^{n-1} and the second contains the rest in ascending order as well, as shown below.

$$[1, 3, \dots, 2^{n-1} – 1] , [2^{n-1} + 1, 2^{n-1} + 3, \dots , 2^n – 1]$$

Then we can say that the first and last number in each subset is the multiplicative inverse of itself modulo 2^n.

Proof:

The proof will be divided into four parts for the four boundary conditions:

$\{1, (2^{n-1} – 1), (2^{n-1} +1)$ and $(2^n – 1)\}$.

a) The proof for 1 as the multiplicative inverse of itself is trivial.
b) The proof for $(2^n – 1)$ as the multiplicative inverse of itself can be given as follows for some integer z that is less than 2^n:
$(2^n– 1) * (2^n – 1) = 2^{2n} - 2^{n+1} + 1 = 2^n (2^n -2) + 1 = 2^n (z) + 1 = 1 \bmod (2^n)$
c) The proof for $(2^{n-1} – 1)$ is simple:
$(2^{n-1} – 1)*(2^{n-1} – 1) = 2^{2n-2} – 2^n + 1 = 2^n * (2^{n-2} – 1) +1 = 1 \bmod (2^n)$
d) The proof for $(2^{n-1} + 1)$ is also simple:
$(2^{n-1} + 1)*(2^{n-1} + 1) = 2^{2n-2} + 2^n + 1 = 2^n * (2^{n-2} + 1) +1 = 1 \bmod (2^n)$

We will also show that for any power of 2, there are only four numbers that are inverses of themselves modulo the power of 2.

Corollary 1

There are exactly four numbers that are the inverses of themselves modulo a power of 2.

Proof

For any number 2^n we proved in Lemma 1 that each of the numbers in the set $S = \{1, (2^{n-1}-1), (2^{n-1}+1), (2^n -1)\}$ equals its inverses modulo (2^n). We need to show that if a number "a" equals its inverse then "a" must be equal to one of the numbers in the set S. Let us assume "a" to be the inverse of itself and that the value of "a" is not equal to any of the elements in the set S, we conclude the following four inequalities:

(i) $a < (2^n -1)$
(ii) $a > 1$
(iii) a is not equal to $(2^{n-1} – 1)$
(iv) a is not equal to $(2^{n-1} +1)$

We will prove that if "a" satisfies conditions (i) and (ii), then "a" must be equal to $(2^{n-1} – 1)$ or $(2^{n-1} +1)$, thereby contradicting statements (iii) and (iv) and proving that the value of "a" can only be equal to one of the members of the set S.

Case (i) $a < (2^n -1)$

Therefore: $a = (2^n – 1) – 2*r$

$$\text{for any } r > 0 \quad (4)$$

and $a^2 = (2^n – 1)^2 – 4 *r (2^n – 1) + 4* r^2$

$a^2 = 2^{2n} – 2^{n+1} + 1 – 4*r*2^n + 4*r+ 4 * r^2$

$a^2 = 2^n (2^n – 2 – 4*r)$

$$+ 4 * r^2 + 4*r + 1 \quad (5)$$

Given the fact that a^2 is congruent to 1 modulo 2^n equation (5) implies that:

$$(4 * r^2 + 4*r) \bmod (2^n) = 0$$

Therefore $(4 * r^2 + 4*r) = t * 2^n$

for $0 < t < 2^n$

Rearranging the terms results in equation (6)

$$r*\{(r+1)/t\}=2^{n-2} \quad (6)$$

Equation (6) implies that both "r" and "{(r+1)/t}" are powers of 2. Since the numerator (r+1) must be an odd number, it follows that {(r+1)/t} must be equal to 1. Therefore "r" will be equal to 2^{n-2}. Consequently, using equation (4), "a" will satisfy the following equation:

$$a = (2^n – 1) – 2*r$$
$$= (2^n – 1) – 2*2^{n-2}$$
$$= (2^{n-1} – 1)$$

Case (ii) $a > 1$

Therefore $a = 1 + 2*r$

for any $r > 0$ (7)

and $a^2 = 1 + 4*r + 4 * r^2$

since "a" is congruent to 1 modulo 2^n it follows that:

$(4 * r^2 + 4*r) \bmod (2^n) = 0$

and eventually we find the value of "r" to be equal to 2^{n-2} following the same steps in *case (i,*. Plugging this value in equation (7) will lead to:

$a = 1 + 2 * 2^{n-2} = (2^{n-1} + 1)$

which completes the proof.

This ends the proof for Corollary 1.

Since mapping of the boundary conditions results in numbers that are inverses of themselves, AM(2^n), and for that matter GF(2^n), may look vulnerable through the S-Box Values. We used corollary 1 to show that regardless of the value of n, the number of integers which are equal to their inverses modulo (2^n) will be exactly four. However, that is not the only concern we have with this approach. AM(2^n) also suffers from another disadvantage, that is, even numbers have no inverses modulo(2^n). The question that may be asked is whether or not modifying the modulo (2^n) arithmetic will lead to a better approach. In the next subsection AM(2^n +1) will be examined.

3.2 Computing inverses modulo ($2^n + 1$)

As stated in the previous subsection, AM(2^n) results in exactly four integers that are equal to their inverses and all the even numbers have no inverses. A reasonable approach is to try only modulo odd integers because modulo even integers will always limit the number of multiplicative inverses as shown in the case of AM(2^n). An interesting odd number is (2^n +1), which is chosen as the nearest odd number greater than 2^n. Figure 1 below shows plotting of three curves for: the powers of 2; AM(2^n); and AM(2^n +1).

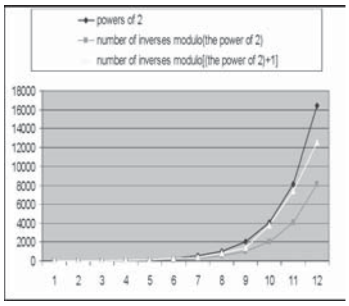

Figure 1. Plotting 2^n, AM(2^n) and AM(2^n +1)

It is clear from the figure that the number of multiplicative inverses under AM(2^n) is equal to 2^{n-1} for all values of n as expected. The figure also shows the number of multiplicative inverses for AM(2^n+1) to be as high as (2^n – 1). The following lemma gives both upper and lower limits for AM(2^n+1).

Lemma 3

The number 'r' of integers that are relatively prime to (2^n+1) is given by the relationship

$2^{n-1} < r < (2^n +1)$

Proof

The proof for the upper bound is obvious since (2^n+1) can be equal to a prime number. The proof that all numbers less than a prime number are relatively prime to that prime number is obvious, hence, the maximum value for 'r' is 2^n. To prove the lower bound, consider expressing (2^n+1) by the product of its prime factors as follows:

$(2^n+1) =$

$p_1^{k} * p_2^{k2} * \ldots * p_m^{km} > 2^n$ (8)

where "m" is less than "n" .

Rewriting equation 4a we get:

$p_1^{k1} * p_2^{k2} * \ldots * p_m^{km} > 2^{L1} * 2^{L2} * \ldots . * 2^{Lm}$ (9)

where $L1+L2+ \ldots +Lm = n$

Each side of inequality (9) has m terms. Since every p_i (i = 1, …, m) in the left hand side is greater than, or equal to 2, and all of the p_i's cannot be equal to 2, then applying Euler's Totiet, function (equation 8) to the left hand side of equation (9) we get:

$[(p_1 – 1) * p_1^{k1-1}] * [(p_2 – 1) * p_2^{k2-1}] * \ldots * [(p_m – 1) * p_m^{km-1}] > 2^{n-1}$ (10)

This completes the proof.

Simulation results also confirm the lower bound showing the number AM(2^n+1) consistently higher than AM(2^n) for all value of n. AM(2^n+1) would have qualified for S-Box computation had it not been for one problem. That is, the distance between two consecutive powers of 2 increases exponentially making both AM(2^n) and AM(2^n+1) less attractive. Another possibility is to use a Brute Force (BF) method in finding a class of reasonably distributed numbers. Each of these numbers should be Relatively Prime (RP) to most of the integers that are less that it. In the next subsection we will discuss the BF approach.

3.3 Computing inverses modulo a number using BF method

The plotting on figure 2 shows the distribution of relatively prime numbers for each of the numbers from 1 to 200.

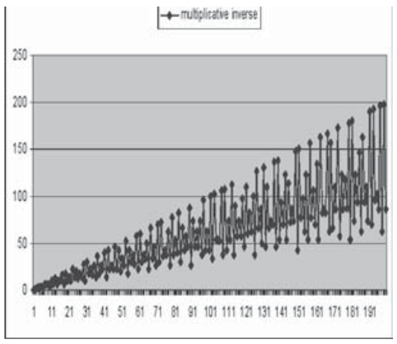

Figure 2. Multiplicative inverses for numbers between 1 and 200

One interesting observation is that the lowest points (minima) on the graph are of the form (3*m) and the highest points (maxima) are of the form (6*m +1) or (6*m-1). Although the graph shows only the plotting for number from 1 to 200, the observation is valid for larger numbers as well. The number of RP integers for a maxima point “M” is, in most cases, equal to “M-1” but always greater than “M/2”. A closer look at the maxima points on the graph reveals that each takes one of the two forms { (6*m +1) or (6*m-1) where m is an integer. Either of the forms is in general equals to a prime number. In the next subsection arithmetic modulo a prime number, AM(P), will be discussed.

3.4 Computing inverses using AM(P).

Since every number less than a prime number P is relatively prime to P, it follows that every number less than P has a multiplicative inverse modulo P. Such is the case because the set of modulo a prime numbers define a field, a known mathematical set with the property that every element in it has a multiplicative inverse.

An obvious concern is that going for the less complex AM(P) may render cryptanalysis easier! The answer to this concern is given in the next subsection.

3.5 Security of the AM(P) approach.

Since the only change in the modified AES is the replacement of the GF with AM(P), it is sufficient to show that the use of AM(P) will not adversely affect security. Given the fact that the inverse of an integer Q in GF is equal to the remainder when Q is divided by a specific number, it is clear that the degree of difficulty of finding inverses in GF is relatively comparable to that of finding inverses using AM(P). Nevertheless, the approach that is taken for the purpose of this research adds more confusion to the process. One can therefore argue that the resulting AES would be more secured than the original Rijndael proposal. The tradeoff for extra security will be the addition of one bit to the size of the S-Box entry. To illustrate this fact, the output of a Java Script program in appendix ‘a’ is given in table 3 below for a prime number 19 on the second row. The first column shows the prime numbers 19, 23, 29 and 31 as row headers. The first row shows the numbers 2 through 18 as column headers. Each entry in the second row shows the multiplicative inverse for the column header obtained using modulo 19 (the row header). The choice of small numbers here is only for simplicity of illustration.

	19	23	29	31
2	10	12	15	16
3	13	8	10	21
4	5	6	22	8
5	4	14	6	25
6	16	4	5	26
7	11	10	25	9
8	12	3	11	4
9	17	18	13	7
10	2	7	25	28
11	7	21	8	17
12	8	2	17	13
13	3	16	9	12
14	15	5	27	20
15	14	20	2	29
16	6	13	20	2
17	9	19	12	11
18	18	9	21	19

Table 3. Inverses of the numbers 2-18 using AM(P) for P = 19, 23, 29 and 31

In general, the process of finding inverses using AM(P) gives a mapping that can be defined as one-to-one onto. This may appear as a weakness for AM(P) and for that matter it may also be thought of as a weakness for GF. How can AM(P) then be made more secure? The answer is simple, by adding more confusion. That is, instead of considering only the prime number P for computing the inverses of the numbers 2 through (P-1), one can compute the inverses for the same numbers but modulo any prime number that is greater than P. To further clarify this concept, rows 3, 4, and 5 of table 3 show the inverses of the numbers 2 through 18 using modulo 23, 29 and 31 respectively. It can easily be seen that the inverses for the same number modulo different prime numbers are different. For example, table 3 shows the inverses for the number 11 to be 7, 21, 8 and 17 in modulo 19, modulo 23, modulo 29 and modulo 31 respectively. The only restriction here is that the prime number chosen should not result in inverses that require more than one extra bit to encode.

4. Conclusion

An implementation of Advanced Encryption Standard that uses the field of prime numbers instead of the Galois Field originally proposed by Rijndael has been investigated. The former is simpler and requires less execution time and implementation circuitry compared to the latter. The strategy for the proposed modulo prime arithmetic adds more confusion to the S-Box computation by allowing the user to choose a prime number from a set of possible candidates. It also adds security by allowing the computation of the S-Box to take place on real time basis. Testing and comparison of simulation performances of the inverse function of S-Box computation using modulo a power-of-two, modulo a power-of-two plus one, and modulo a prime number reveals that the modulo a prime number approach has the best performance. The proposed implementation will provide a better alternative that can be embedded in many systems.

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, Ian (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., Rijmen, V (2005). AES Proposal: Rijndael, Document vers on 2, Date: 03/09/99. Retrieved on October 20, 2005.

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

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

[6] Chodowiec, Pawel., Gaj, Kris (2002). Comparison of the Hardware Performance of AES candidates using reconfigurable hardware.

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

[8] A. Satoh, A., Morioka, S., Takano, K., Munetoh, S (2001). A Compact Rijndael Hardware Architecture with S-Box Optimization, *In*: Proc.Advances in Cryptology—ASIACRYPT 2001, p. 239-254.

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

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

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

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

[13] B. Schneier,B., Kelsey, B., Whiting, J., Wagner, D., Hall, D., Ferguson, N, (1998). **"**Twofish: A 128-Bit Block Cipher".

[14] Ross Anderson, Ross., Biham, Eli., Knudsen, Lars (2000). The Case for Serpent.

[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 (2000). MARS and the AES Selection Criteria.

[17] Daemen, J., Rijmen, V. "AES Proposal: Rijndael" Document version 2

Appendix A

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- saved from url=(0085)file://C:\Documents and Settings\user\Desktop\calc_PrimeFactors\calc_PrimeFactors.htm -->
<HTML><HEAD>
<META http-equiv=Content-Type content="text/html; charset=windows-1256">
<STYLE>.clr {
FONT-SIZE: 8pt; FONT-FAMILY: verdana, Arial, sans-serif }
</STYLE>
<SCRIPT language=VBSCRIPT>
sub cmdcalc_onclick()
   Dim inival, pcnt, retval, totval, pval, retstr
   Dim parr()
   pval = Cint(txtval.value)
   if pval <= 0 then
      Msgbox "Please enter a valid Prime Number"
      exit sub
   end if
   inival = pval - 1
   ReDim parr(inival, 2)
   For pcnt = inival To 2 Step -1
      For i = pcnt To 2 Step -1
         totval = (pcnt * i) - 1
            If (totval Mod pval) = 0 Then
                  parr(pcnt, 0) = pcnt
                  parr(pcnt, 1) = i
            End If
      Next
Next
   For i = inival To 2 Step -1
            if parr(i, 0) = "" then
               parr(i, 0) = i
               parr(i, 1) = GetFactor(parr, inival, i)
            end if
            if parr(i, 0) <> 0 then
               retval = retval + CStr(parr(i, 0)) + " <-> "
             + CStr(parr(i, 1)) + "|"
            end if
  Next
   if retval <> "" then
      lblText.innerHTML = "<B> Valid factors for the
      <BR> Prime Number " + Cstr(pval) + "
        are:</b><br>"
       lblresult.innerhtml = cstr(retval)
   end if
end sub
function GetFactor(parr, inival, srcval)
```

```
   dim retval
   For i = inival To 2 Step -1
        if parr(i, 1) = srcval then
             GetFactor = i
             exit function
        end if
   Next
end function

</SCRIPT>
<META content="MSHTML 6.00.2900.2802" name=GENERATOR></HEAD>
<BODY bgColor=#ffff99>
<H2>
<CENTER>Valid Factors for Prime Numbers </CENTER></H2>
<TABLE>
 <TBODY>
 <TR>
    <TD><LABEL class=clr>Please Enter a valid Prime Number:</LABEL> </TD>
  <TD><INPUT class=clr size=3 value=0 name=txtval> </TD>
    <TD><INPUT class=clr type=button value=Calculate name=cmdcalc> </TD></TR>
 <TR>
    <TD align=middle colSpan=3 height=10><LABEL class=clr></LABEL></TD></TR>
 <TR>
   <TD vAlign=top align=left colSpan=2><LABEL class=clr
id=lblText></LABEL></TD>
   <TD align=left><LABEL class=clr
id=lblresult></LABEL></TD></TR></TBODY></TABLE></BODY></HTML>
```