PNNL-14170

Pacific Northwest
National Laboratory
Operated by Battelle for the
U.S. Department of Energy

Implementation of the AES as a Hash Function for Confirming the Identity of Software on a Computer System

R.R. Hansen
R.B. Bass
R.T. Kouzes
N.D. Mileson

January 2003

Prepared for the U.S. Department of Energy
under Contract DE-AC06-76RL01830

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor Battelle Memorial Institute, nor any of their employees, makes **any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights**. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-ACO6-76RL0183O

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service,
U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161
ph: (800) 553-6847
fax: (703) 605-6900
email: orders@ntis.fedworld.gov
online ordering: http://www.ntis.gov/ordering.htm


This document was printed on recycled paper.
(8/00)

PNNL-14170

Implementation of the AES as a Hash Function for Confirming the Identity of Software on a Computer System

Randy Hansen
Bob Bass
Richard Kouzes
Nick Mileson

January 15, 2003

Prepared for the U.S. Department of Defense
Defense Threat Reduction Agency

Pacific Northwest National Laboratory
Richland, Washington 99352

Implementation of the AES as a Hash Function for Confirming the Identity of Software on a Computer System

1 INTRODUCTION

This paper provides a brief overview of the implementation of the Advanced Encryption Standard (AES) as a hash function for confirming the identity of software resident on a computer system.

The PNNL Software Authentication team chose to use a hash function to confirm software identity on a system for situations where: (1) there is limited time to perform the confirmation and (2) access to the system is restricted to keyboard or thumbwheel input and output can only be displayed on a monitor.

PNNL reviewed three popular algorithms: the Secure Hash Algorithm – 1 (SHA-1), the Message Digest – 5 (MD-5), and the Advanced Encryption Standard (AES) and selected the AES to incorporate in software confirmation tool we developed. This paper gives a brief overview of the SHA-1, MD-5, and the AES and sites references for further detail. It then explains the overall processing steps of the AES to reduce a large amount of generic data—the plain text, such is present in memory and other data storage media in a computer system, to a small amount of data—the hash digest, which is a mathematically unique representation or signature of the former that could be displayed on a computer's monitor.

This paper starts with a simple definition and example to illustrate the use of a hash function. It concludes with a description of how the software confirmation tool uses the hash function to confirm the identity of software on a computer system.

2 DEFINITION AND EXAMPLE OF THE USE OF HASH FUNCTIONS

In general a hash function involves the process of taking some data, efficiently reducing the data into a unique code, and, if necessary, saving that code in specified memory. The Federal Information Processing Standards Publication (FIPS) 197 defines a Hash Function as follows:

> *An approved mathematical function that maps a string of arbitrary length (up to a pre-determined maximum size) to a fixed length string. It may be used to produce a checksum, called a hash value or message digest, for a potentially long string or message.* (Federal Information Processing Standards Publication 197, page 2)

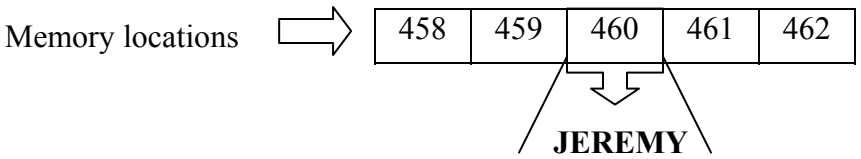
A simple example is taking a name, reducing the name to a single letter or number, and storing the smaller name in memory. There are many ways to reduce data into a unique code, each way with its own degree of complexity and efficiency. The following is a possible implementation of a hash function:

- *Take some data – like this name:* **JEREMY**
- *Now break up the name into some unique code:*

  NOTE: In this example, the letters will be changed into the corresponding ASCII number and then the numbers will be added together.

  J + E + R + E + M + Y

  74 + 69 + 82 + 69 + 77 + 89 = 460
- *The name **JEREMY** has the unique code of 460 for this example. If necessary, The code 460 can be later identified as the name JEREMY by looking in a memory location addressed 460:*



Example 1: *Simple Hash Example*

3 ALGORITHM OVERVIEW

3.1 Secure Hash Algorithm – 1 (SHA-1)

The SHA-1 is a hash algorithm designed to be used with the Digital Signature Algorithm (DSA) to generate or verify a signature for electronic mail, electronic funds transfer, software distribution, data storage, and other applications that require data integrity assurance and data origin authentication.

The SHA-1 is closely modeled after the Message Digest - 4 (MD-4.) For messages less than 2^{64} bits, the algorithm produces a 160-bit signature of the original message.

With enough time and money, the SHA-1 can be conquered by hackers. It isn't 100 percent secure. In addition, there will always be a compromise in speed verses security. The larger (in bits) the message digest is, the more secure the transfer but the slower the speed. A disadvantage might arise with slow and overprotected data or with fast and under protected data.

For more information on the SHA-1 see the following reference:

http://www.itl.nist.gov/fipspubs/fip180-1.htm

3.2 Message Digest – 5 (MD-5)

"[The MD5 algorithm] takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be

"compressed" in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA."

Though the MD-5 is a strong algorithm, it is not as internationally recognized as the AES.

For more information on the MD-5, see the following reference:

http://rfc.sunsite.dk/rfc/rfc1321.html

3.3 Rijndael Algorithm [Advanced Encryption Standard (AES)]

The Rijndael algorithm is the algorithm selected by NIST as the Advanced Encryption Standard (AES). The AES is a new Federal Information Processing Standard (FIPS) Publication that specifies a cryptographic algorithm for use by U.S. Government organizations to protect sensitive (unclassified) information.

Rijndael is reported to be a very good performer in both hardware and software across a wide range of computing environments. It is relatively easy to set up, and it is flexible in terms of block and key size. Rijndael also has low memory requirements, which makes it suitable as a on-board application where memory is restricted.

In its selection of the Rijndael algorithm as the AES, NIST was looking for an algorithm that could be widely used on a voluntary basis by organizations, institutions, and individuals outside of the U.S. Government - and outside of the United States.

For more information of the AES, see the following reference:

http://csrc.nist.gov/CryptoToolkit/aes/

4 PROCESSING STEPS FOR THE AES

The basic steps of AES algorithm are given here. The steps process "plain text"; however, for software confirmation this "plain text" is the binary representation of software and data of interest within a target computer system. The process produces a "hash digest" that represents a unique and very condensed signature of the "plain text".

The process uses four main components that are built into the algorithm or entered as input – the plain text (input), a user key (input), a fixed polynomial (defined within the algorithm), and the S-Box Matrix (actually generated by the algorithm) to generate a hash digest. The key and hash digest can be independently specified at 128, 192, or 256 bits. The steps that follow assume 128 bits for each.

Step 1 Generate the initial key matrix by creating and extracting from the key schedule[1]. Then generate the S-Box Matrix[2] within the algorithm for later use.

[1] Appendix A discusses Key Expansion for a 128-bit key

Step 2 Transpose the Key Matrix to match the column-oriented State Matrix.

$$\begin{bmatrix} 00 & 01 & 02 & 03 \\ 04 & 05 & 06 & 07 \\ 08 & 09 & 0A & 0B \\ 0C & 0D & 0E & 0F \end{bmatrix}' = \begin{bmatrix} 00 & 04 & 08 & 0C \\ 01 & 05 & 09 & 0D \\ 02 & 06 & 0A & 0E \\ 03 & 07 & 0B & 0F \end{bmatrix}$$

Key Matrix — Transposed Key Matrix

Equation 1: Transpose Key Matrix to Match Column-Oriented State Matrix

Step 3 Retrieve the first (or next) 16 bytes of the plain text and reshape into a matrix with four rows and four columns. This matrix is the State Matrix.

$$00\ 11\ 22\ 33\ 44\ 55\ 66\ 77\ 88\ 99\ AA\ BB\ CC\ DD\ EE\ FF \Rightarrow \begin{bmatrix} 00 & 44 & 88 & CC \\ 11 & 55 & 99 & DD \\ 22 & 66 & AA & EE \\ 33 & 77 & BB & FF \end{bmatrix}$$

State Matrix

Equation 2: Reshaping the 16 bytes (128 bits) of "Plain Text" into a 4 x 4 Matrix

Step 4 Bit-wise XOR[3] the State Matrix and the Transposed Key Matrix to form a New State Matrix.

$$\begin{bmatrix} 00 & 44 & 88 & CC \\ 11 & 55 & 99 & DD \\ 22 & 66 & AA & EE \\ 33 & 77 & BB & FF \end{bmatrix} \oplus \begin{bmatrix} 00 & 04 & 08 & 0C \\ 01 & 05 & 09 & 0D \\ 02 & 06 & 0A & 0E \\ 03 & 07 & 0B & 0F \end{bmatrix} = \begin{bmatrix} 00 & 40 & 80 & C0 \\ 10 & 50 & 90 & D0 \\ 20 & 60 & A0 & E0 \\ 30 & 70 & B0 & F0 \end{bmatrix}$$

State Matrix — Transposed Key Matrix — New State Matrix

Equation 3: Bit-wise XOR of State Matrix and Key Matrix

---

[2] Appendix B discusses the formation of the S-Box Matrix

[3] Appendix C shows an example of a Bit-wise XOR.

Step 5 Perform a Round Transformation

5.a Replace each element of New State Matrix with an element from the S-Box Matrix[4]. Using each value in the New State Matrix as an address into the S-Box Matrix, substitute the value at the corresponding address in the S-Box Matrix into the State Matrix After Substitution

New State Matrix

$$\begin{bmatrix} xx & xx & xx & xx \\ xx & 50 & xx & xx \\ xx & xx & xx & xx \\ xx & xx & xx & xx \end{bmatrix}$$

$\Rightarrow$

State Matrix After Substitution

$$\begin{bmatrix} xx & xx & xx & xx \\ xx & 53 & xx & xx \\ xx & xx & xx & xx \\ xx & xx & xx & xx \end{bmatrix}$$

S - Box Matrix

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
1	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
2	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
3	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
4	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
5	53	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
6	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
7	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
8	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
9	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
A	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
B	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
C	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
D	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
E	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
F	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·

Equation 4: Replacement of One Element of State Matrix by Appropriate S-Box Element

For example:

[4] Appendix B discusses the formation of the S-Box Matrix.

New State Matrix

$$\begin{bmatrix} 00 & 40 & 80 & C0 \\ 10 & 50 & 90 & D0 \\ 20 & 60 & A0 & E0 \\ 30 & 70 & B0 & F0 \end{bmatrix}$$

$\Rightarrow$

State Matrix After Substitution

$$\begin{bmatrix} 63 & 09 & CD & BA \\ CA & 53 & 60 & 70 \\ B7 & D0 & E0 & E1 \\ 04 & 51 & E7 & 8C \end{bmatrix}$$

S - Box Matrix

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

Example 2: Substitution of S-Box Matrix Elements into the State Matrix After Substitution

5.b Shift the rows of the State Matrix After Substitution to the left to form the Left-Shifted State Matrix. Each row is shifted an additional position to the left: the first row is not shifted, the second row is shifted one position to the left, the third row is shifted two positions to the left, and the fourth row is shifted three positions to the left.

State Matrix After Substitution

$$\begin{bmatrix} 63 & 09 & CD & BA \\ CA & 53 & 60 & 70 \\ B7 & D0 & E0 & E1 \\ 04 & 51 & E7 & 8C \end{bmatrix}$$

$\Rightarrow$

Left - Shifted State Matrix

$$\begin{bmatrix} 63 & 09 & CD & BA \\ 53 & 60 & 70 & CA \\ E0 & E1 & B7 & D0 \\ 8C & 04 & 51 & E7 \end{bmatrix}$$

Equation 5: Shifting Rows to the Left.

5.c Apply a MixColumns Transformation to the Left-Shifted State Matrix to produce the B State Matrix. Left multiplying the Left-Shifted State Matrix by a Fixed Polynomial (P) accomplishes a MixColumns Transformation.

$$\begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \\ B_{31} & B_{32} & B_{33} & B_{34} \\ B_{41} & B_{42} & B_{43} & B_{44} \end{bmatrix} = P \bullet S \text{, where } B_{ij} = \begin{bmatrix} P_{i1} & P_{i2} & P_{i3} & P_{i4} \end{bmatrix} \bullet \begin{bmatrix} S_{1j} \\ S_{2j} \\ S_{2j} \\ S_{2j} \end{bmatrix}$$

$$B_{ij} = (P_{i1} \circ S_{1j}) \oplus (P_{i2} \circ S_{2j}) \oplus (P_{i3} \circ S_{3j}) \oplus (P_{i4} \circ S_{4j}),$$

where :

$\circ$ = binary polynomial modulo multiplication

$\oplus$ = bit - wise xor

Equation 6: Left Multiplication of the State Matrix by Fix Polynomial Matrix (P).

Fixed Poly Matrix Left - Shifted State Matrix

$$\begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \\ B_{31} & B_{32} & B_{33} & B_{34} \\ B_{41} & B_{42} & B_{43} & B_{44} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 20 & 30 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} 63 & 09 & CD & BA \\ 53 & 60 & 70 & CA \\ E0 & E1 & B7 & D0 \\ 8C & 04 & 51 & E7 \end{bmatrix}$$

$$\begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \\ B_{31} & B_{32} & B_{33} & B_{34} \\ B_{41} & B_{42} & B_{43} & B_{44} \end{bmatrix} = \begin{bmatrix} 5F & 57 & F7 & 1D \\ 72 & F5 & BE & B9 \\ 64 & BC & 3B & F9 \\ 15 & 92 & 29 & 1A \end{bmatrix},$$

B State Matrix

where :

$$B_{22} = \begin{bmatrix} 01 & 02 & 03 & 01 \end{bmatrix} \bullet \begin{bmatrix} 09 \\ 60 \\ E1 \\ 04 \end{bmatrix} = (01 * 09) \oplus (02 * 60) \oplus (03 * E1) \oplus (01 * 04) = F5$$

Example 3: B State Matrix after the MixColumn Transformation

5.d Bit-wise XOR the B-State Matrix with the transpose of the Next Key Matrix, which is the next four rows of the Key Schedule, to produce the Round Transformed Matrix (see Appendix A for Key Schedule definition.)

$$\underset{\text{B State Matrix}}{\begin{bmatrix} 5F & 57 & F7 & 1D \\ 72 & F5 & BE & B9 \\ 64 & BC & 3B & F9 \\ 15 & 92 & 29 & 1A \end{bmatrix}} \oplus \underset{\text{Next Key Matrix}}{\begin{bmatrix} D6 & D2 & DA & D6 \\ AA & AF & A6 & AB \\ 74 & 72 & 78 & 76 \\ FD & FA & F1 & FE \end{bmatrix}} = \underset{\text{Round Transformed Matrix}}{\begin{bmatrix} 89 & D8 & 10 & E8 \\ 85 & 5A & CE & 68 \\ 2D & 18 & 43 & D8 \\ CB & 12 & 8F & E4 \end{bmatrix}}$$

Equation 7: Bit-Wise XOR of the B State Matrix and the Next Key Matrix.

Step 6 Repeat the Round Transformation eight more times (Step 5).

Step 7 Perform the S-Box substitution described in Step Step 5 using the Round Transformed Matrix formed in Step 6.

Step 8 Shift the rows of the State Matrix After Substitution from Step 7 as described in Step 5.b .

Step 9 Perform a Bit-wise XOR the outputted matrix from 0 as described in Step 5.d .

The final digest value for the entire plain text is generated by repeating 0- Step 8 on the remaining plain text, 16 bytes at a time. After Step 8 the resulting 16-byte, Round-Transformed State Matrix is the cipher text of the 16-byte plain text input. The iterative process through the remaining plain text uses the cipher text of the previous 16-bytes of plan text as the Matrix Key for the next 16 bytes of plain text. The final digest value is the Step 8 cipher text of the last 16 bytes of the plain text.

5 THE SOFTWARE CONFIRMATION TOOL

PNNL development of a multi-purpose software tool that can read any selection from all of a system's addressable memory (RAM or ROM), IO registers, and storage devices (e.g., standard or solid-state disks) and perform a hash on them.

A user can confirm the software on the target computer system by comparing the hash digest the tool generates for a specific user-entered key with the hash digest generated by the same process and the same key on a validated, duplicate system.

Prior to using this tool for software confirmation, it is necessary to map the target computer system to determine the range of addresses (for each of the system's memory devices) that will be input to the tool's hash function. The mapping must identify all the addressable memory locations in RAM, I/O registers, and the storage devices that are expected to remain unchanged (static) when all the code is loaded on the system and the system is performing its functions. It is only the values in the static memory locations that the tool can hash and generate a unique hash digest for a given input key.

The byte values in the selected range of all selected memory devices, taken as a whole, constitute the "plain text" input to the hash function. The tool treats non-static values or ranges of values as a specific constant and includes them in the "plain text" input, or if the ranges are large and multiples of 16 bytes, it can skip them and not include them as input. The tool uses the steps of Section 4 and processes the entire selection, 16 bytes at a time. When the tool completes the hash process, it displays the hash digest on the systems monitor for the user to copy down on a piece of paper or to memorize.

Appendix A Key Expansion For 128-bit key

The AES algorithm takes the original cipher key and uses a key expansion routine to make a key schedule. Upon completion of the key expansion, the resulting key schedule is 4-byte words long.

First, the input 4-byte word key is taken into a routine. The routine applies the key to the S-Box and outputs a new 4-byte word key. The new key is then sent to another function where cyclic permutation is performed on the key – for example:

Key = **[a_0, a_1, a_2, a_3]** *where a_i is 1 word (2 bytes)*

5.3.1.1.1 Key enters cyclic permutation routine

New Key = **[a_1, a_2, a_3, a_0]**

Now the key will enter the key expansion routine to make a key schedule. In the expanded key, the first 4 words are just the cipher key. Each following word is equal to the XOR of the previous word with the word 4 positions earlier – for example, let **w[i]** equal an expanded key word:

Original cipher key → w[1], w[2], w[3], w[4], w[4⊕1], w[(4⊕1)⊕2], etc.

NOTE: All of the above numbers represent indices of w[].

For words whose positions are a multiple of 4, a transformation is completed before the w[i – 1] XOR. Then the word is XORed with predetermined round constant. The transformation consists of a cyclic shift of the bytes and then a table lookup of that byte.

These operations are performed 44 (from w[1] to w[44]) times after the routine for expanding the 128-bit key is completed. (44 is the arbitrarily chosen number for the AES algorithm.) The final expanded key has 44-byte words instead of the original 4-byte words.

(Federal Information Processing Standards Publication 197, page 19-20)

Appendix B S-Box Formation

The S-Box is an invertible matrix composed of two transformations. First, binary polynomial modulo multiplication (as described in appendix D) is used in finding the multiplicative inverse of b(x) where b(x) is any non-zero polynomial. This is done by using the Euclidian Algorithm to find a(x) and c(x):

$$b(x)a(x) + m(x)c(x) = 1$$

where m(x) is a modulo polynomial

This means the inverse of b(x) can be represented by

$$b^{-1}(x) = a(x) \bmod m(x)$$

The element {00} is mapped onto itself in this equation.

Next, the following transformation of b_i is performed:

$$b_i' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

Let $0 \le i < 8$, where b_i is the i^{th} bit of the byte, and c_i is the i^{th} bit byte c with the defined value of {63} hexadecimal or {0110 0011} binary. The actual matrix representation of the equation is as follows:

$$\begin{vmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{vmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{vmatrix} \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{vmatrix}$$

This transformation is performed 256 times until the 16x16, S-Box matrix (found in Example 2: Substitution of S-Box Matrix Elements into the State Matrix After Substitution) is formed.

(Federal Information Processing Standards Publication 197, page 15-16)

Appendix C Example of Bit-Wise XOR

The corresponding elements in the matrices shown below will be bit-wise XORed.

1. The hexadecimal numbers in the matrices must be converted to binary numbers.

$4 = 0100 \therefore 44 = 0100 \cdot 0100$

$$\begin{bmatrix} 00 & 44 & 88 & CC \\ 11 & 55 & 99 & DD \\ 22 & 66 & AA & EE \\ 33 & 77 & BB & FF \end{bmatrix} = \begin{bmatrix} 0000 \cdot 0000 & 0100 \cdot 0100 & 1000 \cdot 1000 & 1100 \cdot 1100 \\ 0001 \cdot 0001 & 0101 \cdot 0101 & 1001 \cdot 1001 & 1101 \cdot 1101 \\ 0010 \cdot 0010 & 0110 \cdot 0110 & 1010 \cdot 1010 & 1110 \cdot 1110 \\ 0011 \cdot 0011 & 0111 \cdot 0111 & 1011 \cdot 1011 & 1111 \cdot 1111 \end{bmatrix}$$

$$\begin{bmatrix} 00 & 04 & 08 & 0C \\ 01 & 05 & 09 & 0D \\ 02 & 06 & 0A & 0E \\ 03 & 07 & 0B & 0F \end{bmatrix} = \begin{bmatrix} 0000 \cdot 0000 & 0000 \cdot 0100 & 0000 \cdot 1000 & 0000 \cdot 1100 \\ 0000 \cdot 0001 & 0000 \cdot 0101 & 0000 \cdot 1001 & 0000 \cdot 1101 \\ 0000 \cdot 0010 & 0000 \cdot 0110 & 0000 \cdot 1010 & 0000 \cdot 1110 \\ 0000 \cdot 0011 & 0000 \cdot 0111 & 0000 \cdot 1011 & 0000 \cdot 1111 \end{bmatrix}$$

2. The binary numbers are bit-wise XORed.

$0000 \cdot 0000$	$0100 \cdot 0100$	$1000 \cdot 1000$	$1100 \cdot 1100$
$\oplus\ 0000 \cdot 0000$	$\oplus\ 0000 \cdot 0100$	$\oplus\ 0000 \cdot 1000$	$\oplus\ 0000 \cdot 1100$
$0000 \cdot 0000$	$0100 \cdot 0000$	$1000 \cdot 0000$	$1100 \cdot 0000$
$0001 \cdot 0001$	$0101 \cdot 0101$	$1001 \cdot 1001$	$1101 \cdot 1101$
$\oplus\ 0000 \cdot 0001$	$\oplus\ 0000 \cdot 0101$	$\oplus\ 0000 \cdot 1001$	$\oplus\ 0000 \cdot 1101$
$0001 \cdot 0000$	$0101 \cdot 0000$	$1001 \cdot 0000$	$1101 \cdot 0000$
$0010 \cdot 0010$	$0110 \cdot 0110$	$1010 \cdot 1010$	$1110 \cdot 1110$
$\oplus\ 0000 \cdot 0010$	$\oplus\ 0000 \cdot 0110$	$\oplus\ 0000 \cdot 1010$	$\oplus\ 0000 \cdot 1110$
$0010 \cdot 0000$	$0110 \cdot 0000$	$1010 \cdot 0000$	$1110 \cdot 0000$
$0011 \cdot 0011$	$0111 \cdot 0111$	$1011 \cdot 1011$	$1111 \cdot 1111$
$\oplus\ 0000 \cdot 0011$	$\oplus\ 0000 \cdot 0111$	$\oplus\ 0000 \cdot 1011$	$\oplus\ 0000 \cdot 1111$
$0011 \cdot 0000$	$0111 \cdot 0000$	$1011 \cdot 0000$	$1111 \cdot 0000$

3. The results are displayed in the final matix.

$$\begin{bmatrix} 00 & 44 & 88 & CC \\ 11 & 55 & 99 & DD \\ 22 & 66 & AA & EE \\ 33 & 77 & BB & FF \end{bmatrix} \oplus \begin{bmatrix} 00 & 04 & 08 & 0C \\ 01 & 05 & 09 & 0D \\ 02 & 06 & 0A & 0E \\ 03 & 07 & 0B & 0F \end{bmatrix} = \begin{bmatrix} 00 & 40 & 80 & C0 \\ 10 & 50 & 90 & D0 \\ 20 & 60 & A0 & E0 \\ 30 & 70 & B0 & F0 \end{bmatrix}$$

Appendix D Binary Polynomial Modulo Multiplication

Example of binary polynomial modulo multiplication:

The modulo equation used in the AES algorithm has to be an irreducible polynomial of degree eight. This means the equation has only 2 divisors – one and itself. The actual equation used is:

$$\mathbf{m(x) = x^8 + x^4 + x^3 + x + 1}$$

or **{01}{1b}** in hexadecimal notation

First, start with two original hexadecimal bytes: **{57}** and **{83}** for example. Next, expand the bytes into their respective polynomials:

{57} hex =

{0101 0111} binary =

$0x^7 + 1x^6 + 0x^5 + 1x^4 + 0x^3 + 1x^2 + 1x^1 + 1x^0 =$

$\mathbf{x^6 + x^4 + x^2 + x + 1}$

{83} hex =

{1000 0011} binary =

$1x^7 + 0x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 1x^1 + 1x^0 =$

$\mathbf{x^7 + x + 1}$

Now multiply the two polynomials together:

$$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^7 +$$
$$x^7 + x^5 + x^3 + x^2 + x +$$
$$x^6 + x^4 + x^2 + x + 1$$
$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

Now use the modulo polynomial to ensure the answer's degree is less than eight so it can be represented by a single byte:

$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$ modulo $(x^8 + x^4 + x^3 + x + 1) =$

$\mathbf{x^7 + x^6 + 1}$ or **{c1}**

(Federal Information Processing Standards Publication 197, page 10-11)

Reference Page

Federal Information Processing Standards Publication 180, "Announcing the Standard for Secure Hash Standard"; http://www.itl.nist.gov/fipspubs/fip180-1.htm, April 17, 1995.

Federal Information Processing Standards Publication 197, "Announcing the Advanced Encryption Standard (AES)"; http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, November 26, 2001.

Rivest, R., "The MD-5 Message Digest Algorithm";http://rfc.sunsite.dk/rfc/rfc1321.html, April 1992.