Federal Information
Processing Standards Publication 197

November 26, 2001

Announcing the

ADVANCED ENCRYPTION STANDARD (AES)

Federal Information Processing Standards Publications (FIPS PUBS) are issued by the National Institute of Standards and Technology (NIST) after approval by the Secretary of Commerce pursuant to Section 5131 of the Information Technology Management Reform Act of 1996 (Public Law 104-106) and the Computer Security Act of 1987 (Public Law 100-235).


3. Explanation. The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information. Encryption converts data to an unintelligible form called ciphertext; decrypting the ciphertext converts the data back into its original form, called plaintext.

The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits.

4. Approving Authority. Secretary of Commerce.


6. Applicability. This standard may be used by Federal departments and agencies when an agency determines that sensitive (unclassified) information (as defined in P. L. 100-235) requires cryptographic protection.

Other FIPS-approved cryptographic algorithms may be used in addition to, or in lieu of, this standard. Federal agencies or departments that use cryptographic devices for protecting classified information can use those devices for protecting sensitive (unclassified) information in lieu of this standard.

In addition, this standard may be adopted and used by non-Federal Government organizations. Such use is encouraged when it provides the desired security for commercial and private organizations.

8. **Implementations.** The algorithm specified in this standard may be implemented in software, firmware, hardware, or any combination thereof. The specific implementation may depend on several factors such as the application, the environment, the technology used, etc. The algorithm shall be used in conjunction with a FIPS approved or NIST recommended mode of operation. Object Identifiers (OIDs) and any associated parameters for AES used in these modes are available at the Computer Security Objects Register (CSOR), located at [http://csrc.nist.gov/csor/][2].

Implementations of the algorithm that are tested by an accredited laboratory and validated will be considered as complying with this standard. Since cryptographic security depends on many factors besides the correct implementation of an encryption algorithm, Federal Government employees, and others, should also refer to NIST Special Publication 800-21, *Guideline for Implementing Cryptography in the Federal Government*, for additional information and guidance (NIST SP 800-21 is available at [http://csrc.nist.gov/publications/]).

9. **Implementation Schedule.** This standard becomes effective on May 26, 2002.

10. **Patents.** Implementations of the algorithm specified in this standard may be covered by U.S. and foreign patents.

11. **Export Control.** Certain cryptographic devices and technical data regarding them are subject to Federal export controls. Exports of cryptographic modules implementing this standard and technical data regarding them must comply with these Federal regulations and be licensed by the Bureau of Export Administration of the U.S. Department of Commerce. Applicable Federal government export controls are specified in Title 15, Code of Federal Regulations (CFR) Part 740.17; Title 15, CFR Part 742; and Title 15, CFR Part 774, Category 5, Part 2.

12. **Qualifications.** NIST will continue to follow developments in the analysis of the AES algorithm. As with its other cryptographic algorithm standards, NIST will formally reevaluate this standard every five years.

Both this standard and possible threats reducing the security provided through the use of this standard will undergo review by NIST as appropriate, taking into account newly available analysis and technology. In addition, the awareness of any breakthrough in technology or any mathematical weakness of the algorithm will cause NIST to reevaluate this standard and provide necessary revisions.

13. **Waiver Procedure.** Under certain exceptional circumstances, the heads of Federal agencies, or their delegates, may approve waivers to Federal Information Processing Standards (FIPS). The heads of such agencies may redelegate such authority only to a senior official designated pursuant to Section 3506(b) of Title 44, U.S. Code. Waivers shall be granted only when compliance with this standard would

   a. adversely affect the accomplishment of the mission of an operator of Federal computer system or
   
   b. cause a major adverse financial impact on the operator that is not offset by government-wide savings.
Agency heads may act upon a written waiver request containing the information detailed above. Agency heads may also act without a written waiver request when they determine that conditions for meeting the standard cannot be met. Agency heads may approve waivers only by a written decision that explains the basis on which the agency head made the required finding(s). A copy of each such decision, with procurement sensitive or classified portions clearly identified, shall be sent to: National Institute of Standards and Technology; ATTN: FIPS Waiver Decision, Information Technology Laboratory, 100 Bureau Drive, Stop 8900, Gaithersburg, MD 20899-8900.

In addition, notice of each waiver granted and each delegation of authority to approve waivers shall be sent promptly to the Committee on Government Operations of the House of Representatives and the Committee on Government Affairs of the Senate and shall be published promptly in the Federal Register.

When the determination on a waiver applies to the procurement of equipment and/or services, a notice of the waiver determination must be published in the Commerce Business Daily as a part of the notice of solicitation for offers of an acquisition or, if the waiver determination is made after that notice is published, by amendment to such notice.

A copy of the waiver, any supporting documents, the document approving the waiver and any supporting and accompanying documents, with such deletions as the agency is authorized and decides to make under Section 552(b) of Title 5, U.S. Code, shall be part of the procurement documentation and retained by the agency.

14. Where to obtain copies. This publication is available electronically by accessing http://csrc.nist.gov/publications/. A list of other available computer security publications, including ordering information, can be obtained from NIST Publications List 91, which is available at the same web site. Alternatively, copies of NIST computer security publications are available from: National Technical Information Service (NTIS), 5285 Port Royal Road, Springfield, VA 22161.
Federal Information
Processing Standards Publication 197

November 26, 2001

Specification for the

ADVANCED ENCRYPTION STANDARD (AES)

Table of Contents

1. INTRODUCTION ................................................................. 5

2. DEFINITIONS ....................................................................... 5
   2.1 GLOSSARY OF TERMS AND ACRONYMS ......................... 5
   2.2 ALGORITHM PARAMETERS, SYMBOLS, AND FUNCTIONS .... 6

3. NOTATION AND CONVENTIONS ....................................... 7
   3.1 INPUTS AND OUTPUTS .................................................. 7
   3.2 BYTES .......................................................................... 8
   3.3 ARRAYS OF BYTES ..................................................... 8
   3.4 THE STATE .................................................................... 9
   3.5 THE STATE AS AN ARRAY OF COLUMNS ....................... 10

4. MATHEMATICAL PRELIMINARIES ................................ 10
   4.1 ADDITION ..................................................................... 10
   4.2 MULTIPLICATION ....................................................... 10
      4.2.1 Multiplication by $x$ ............................................... 11
   4.3 POLYNOMIALS WITH COEFFICIENTS IN $GF(2^8)$ ............ 12

5. ALGORITHM SPECIFICATION ........................................... 13
   5.1 CIPHER ......................................................................... 14
      5.1.1 SubBytes() Transformation ...................................... 15
      5.1.2 ShiftRows() Transformation .................................... 17
      5.1.3 MixColumns() Transformation ................................. 17
      5.1.4 AddRoundKey() Transformation .............................. 18
   5.2 KEY EXPANSION ........................................................... 19
   5.3 INVERSE CIPHER .......................................................... 20
5.3.1 \emph{InvShiftRows()} Transformation

5.3.2 \emph{InvSubBytes()} Transformation

5.3.3 \emph{InvMixColumns()} Transformation

5.3.4 Inverse of the AddRoundKey() Transformation

5.3.5 Equivalent Inverse Cipher

6. IMPLEMENTATION ISSUES

6.1 Key Length Requirements

6.2 Keying Restrictions

6.3 Parameterization of Key Length, Block Size, and Round Number

6.4 Implementation Suggestions Regarding Various Platforms

APPENDIX A - KEY EXPANSION EXAMPLES

A.1 Expansion of a 128-bit Cipher Key

A.2 Expansion of a 192-bit Cipher Key

A.3 Expansion of a 256-bit Cipher Key

APPENDIX B – CIPHER EXAMPLE

APPENDIX C – EXAMPLE VECTORS

C.1 AES-128 (Nk=4, Nr=10)

C.2 AES-192 (Nk=6, Nr=12)

C.3 AES-256 (Nk=8, Nr=14)

APPENDIX D - REFERENCES
Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hexadecimal representation of bit patterns</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Indices for Bytes and Bits</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>State array input and output</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>Key-Block-Round Combinations</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>Pseudo Code for the Cipher</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td><code>SubBytes()</code> applies the S-box to each byte of the State</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>S-box: substitution values for the byte (xy) (in hexadecimal format)</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td><code>ShiftRows()</code> cyclically shifts the last three rows in the State</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td><code>MixColumns()</code> operates on the State column-by-column</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td><code>AddRoundKey()</code> XORs each column of the State with a word from the key schedule</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>Pseudo Code for Key Expansion</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>Pseudo Code for the Inverse Cipher</td>
<td>21</td>
</tr>
<tr>
<td>13</td>
<td><code>InvShiftRows()</code> cyclically shifts the last three rows in the State</td>
<td>22</td>
</tr>
<tr>
<td>14</td>
<td>Inverse S-box: substitution values for the byte (xy) (in hexadecimal format)</td>
<td>22</td>
</tr>
<tr>
<td>15</td>
<td>Pseudo Code for the Equivalent Inverse Cipher</td>
<td>25</td>
</tr>
</tbody>
</table>
1. Introduction

This standard specifies the Rijndael algorithm ([3] and [4]), a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. Rijndael was designed to handle additional block sizes and key lengths, however they are not adopted in this standard.

Throughout the remainder of this standard, the algorithm specified herein will be referred to as “the AES algorithm.” The algorithm may be used with the three different key lengths indicated above, and therefore these different “flavors” may be referred to as “AES-128”, “AES-192”, and “AES-256”.

This specification includes the following sections:

2. Definitions of terms, acronyms, and algorithm parameters, symbols, and functions;
3. Notation and conventions used in the algorithm specification, including the ordering and numbering of bits, bytes, and words;
4. Mathematical properties that are useful in understanding the algorithm;
5. Algorithm specification, covering the key expansion, encryption, and decryption routines;
6. Implementation issues, such as key length support, keying restrictions, and additional block/key/round sizes.

The standard concludes with several appendices that include step-by-step examples for Key Expansion and the Cipher, example vectors for the Cipher and Inverse Cipher, and a list of references.

2. Definitions

2.1 Glossary of Terms and Acronyms

The following definitions are used throughout this standard:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>Affine</td>
<td>A transformation consisting of multiplication by a matrix followed by</td>
</tr>
<tr>
<td>Transformation</td>
<td>the addition of a vector.</td>
</tr>
<tr>
<td>Array</td>
<td>An enumerated collection of identical entities (e.g., an array of bytes).</td>
</tr>
<tr>
<td>Bit</td>
<td>A binary digit having a value of 0 or 1.</td>
</tr>
<tr>
<td>Block</td>
<td>Sequence of binary bits that comprise the input, output, State, and</td>
</tr>
<tr>
<td></td>
<td>Round Key. The length of a sequence is the number of bits it contains.</td>
</tr>
<tr>
<td></td>
<td>Blocks are also interpreted as arrays of bytes.</td>
</tr>
<tr>
<td>Byte</td>
<td>A group of eight bits that is treated either as a single entity or as an</td>
</tr>
<tr>
<td></td>
<td>array of 8 individual bits.</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cipher</td>
<td>Series of transformations that converts plaintext to ciphertext using the Cipher Key.</td>
</tr>
<tr>
<td>Cipher Key</td>
<td>Secret, cryptographic key that is used by the Key Expansion routine to generate a set of Round Keys; can be pictured as a rectangular array of bytes, having four rows and $N_k$ columns.</td>
</tr>
<tr>
<td>Ciphertext</td>
<td>Data output from the Cipher or input to the Inverse Cipher.</td>
</tr>
<tr>
<td>Inverse Cipher</td>
<td>Series of transformations that converts ciphertext to plaintext using the Cipher Key.</td>
</tr>
<tr>
<td>Key Expansion</td>
<td>Routine used to generate a series of Round Keys from the Cipher Key.</td>
</tr>
<tr>
<td>Plaintext</td>
<td>Data input to the Cipher or output from the Inverse Cipher.</td>
</tr>
<tr>
<td>Rijndael</td>
<td>Cryptographic algorithm specified in this Advanced Encryption Standard (AES).</td>
</tr>
<tr>
<td>Round Key</td>
<td>Round keys are values derived from the Cipher Key using the Key Expansion routine; they are applied to the State in the Cipher and Inverse Cipher.</td>
</tr>
<tr>
<td>State</td>
<td>Intermediate Cipher result that can be pictured as a rectangular array of bytes, having four rows and $N_b$ columns.</td>
</tr>
<tr>
<td>S-box</td>
<td>Non-linear substitution table used in several byte substitution transformations and in the Key Expansion routine to perform a one-for-one substitution of a byte value.</td>
</tr>
<tr>
<td>Word</td>
<td>A group of 32 bits that is treated either as a single entity or as an array of 4 bytes.</td>
</tr>
</tbody>
</table>

### 2.2 Algorithm Parameters, Symbols, and Functions

The following algorithm parameters, symbols, and functions are used throughout this standard:

- **AddRoundKey()**: Transformation in the Cipher and Inverse Cipher in which a Round Key is added to the State using an XOR operation. The length of a Round Key equals the size of the State (i.e., for $N_b = 4$, the Round Key length equals 128 bits/16 bytes).

- **InvMixColumns()**: Transformation in the Inverse Cipher that is the inverse of **MixColumns()**.

- **InvShiftRows()**: Transformation in the Inverse Cipher that is the inverse of **ShiftRows()**.

- **InvSubBytes()**: Transformation in the Inverse Cipher that is the inverse of **SubBytes()**.

- **$K$**: Cipher Key.
MixColumns() Transformation in the Cipher that takes all of the columns of the State and mixes their data (independently of one another) to produce new columns.

\textbf{\textit{Nb}} Number of columns (32-bit words) comprising the State. For this standard, \( Nb = 4 \). (Also see Sec. 6.3.)

\textbf{\textit{Nk}} Number of 32-bit words comprising the Cipher Key. For this standard, \( Nk = 4, 6, \) or 8. (Also see Sec. 6.3.)

\textbf{\textit{Nr}} Number of rounds, which is a function of \( Nk \) and \( Nb \) (which is fixed). For this standard, \( Nr = 10, 12, \) or 14. (Also see Sec. 6.3.)

\textbf{\textit{Rcon[ ]}} The round constant word array.

\textbf{\textit{RotWord( )}} Function used in the Key Expansion routine that takes a four-byte word and performs a cyclic permutation.

\textbf{\textit{ShiftRows( )}} Transformation in the Cipher that processes the State by cyclically shifting the last three rows of the State by different offsets.

\textbf{\textit{SubBytes( )}} Transformation in the Cipher that processes the State using a non-linear byte substitution table (S-box) that operates on each of the State bytes independently.

\textbf{\textit{SubWord( )}} Function used in the Key Expansion routine that takes a four-byte input word and applies an S-box to each of the four bytes to produce an output word.

\textbf{\textit{XOR}} Exclusive-OR operation.

\( \oplus \) Exclusive-OR operation.

\( \otimes \) Multiplication of two polynomials (each with degree \(< 4\)) modulo \( x^4 + 1 \).

\( \bullet \) Finite field multiplication.

3. Notation and Conventions

3.1 Inputs and Outputs

The \textit{input} and \textit{output} for the AES algorithm each consist of \textbf{sequences of 128 bits} (digits with values of 0 or 1). These sequences will sometimes be referred to as \textbf{blocks} and the number of bits they contain will be referred to as their length. The \textbf{Cipher Key} for the AES algorithm is a \textbf{sequence of 128, 192 or 256 bits}. Other input, output and Cipher Key lengths are not permitted by this standard.

The bits within such sequences will be numbered starting at zero and ending at one less than the sequence length (block length or key length). The number \( i \) attached to a bit is known as its index and will be in one of the ranges \( 0 \leq i < 128 \), \( 0 \leq i < 192 \) or \( 0 \leq i < 256 \) depending on the block length and key length (specified above).
3.2 Bytes

The basic unit for processing in the AES algorithm is a **byte**, a sequence of eight bits treated as a single entity. The input, output and Cipher Key bit sequences described in Sec. 3.1 are processed as arrays of bytes that are formed by dividing these sequences into groups of eight contiguous bits to form arrays of bytes (see Sec. 3.3). For an input, output or Cipher Key denoted by \( a \), the bytes in the resulting array will be referenced using one of the two forms, \( a_n \) or \( a[n] \), where \( n \) will be in one of the following ranges:

- Key length = 128 bits, \( 0 \leq n < 16 \);
- Block length = 128 bits, \( 0 \leq n < 16 \);
- Key length = 192 bits, \( 0 \leq n < 24 \);
- Key length = 256 bits, \( 0 \leq n < 32 \).

All byte values in the AES algorithm will be presented as the concatenation of its individual bit values (0 or 1) between braces in the order \( \{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\} \). These bytes are interpreted as finite field elements using a polynomial representation:

\[
    b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i x^i.
\]  

(3.1)

For example, \( \{01100011\} \) identifies the specific finite field element \( x^6 + x^5 + 1 \).

It is also convenient to denote byte values using hexadecimal notation with each of two groups of four bits being denoted by a single character as in Fig. 1.

<table>
<thead>
<tr>
<th>Bit Pattern</th>
<th>Character</th>
<th>Bit Pattern</th>
<th>Character</th>
<th>Bit Pattern</th>
<th>Character</th>
<th>Bit Pattern</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0100</td>
<td>4</td>
<td>1000</td>
<td>8</td>
<td>1100</td>
<td>c</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>0101</td>
<td>5</td>
<td>1001</td>
<td>9</td>
<td>1101</td>
<td>d</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>0110</td>
<td>6</td>
<td>1010</td>
<td>a</td>
<td>1110</td>
<td>e</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>0111</td>
<td>7</td>
<td>1011</td>
<td>b</td>
<td>1111</td>
<td>f</td>
</tr>
</tbody>
</table>

**Figure 1.** Hexadecimal representation of bit patterns.

Hence the element \( \{01100011\} \) can be represented as \( \{63\} \), where the character denoting the four-bit group containing the higher numbered bits is again to the left.

Some finite field operations involve one additional bit (\( b_8 \)) to the left of an 8-bit byte. Where this extra bit is present, it will appear as ‘\{01\}’ immediately preceding the 8-bit byte; for example, a 9-bit sequence will be presented as \( \{01\}1b \).

3.3 Arrays of Bytes

Arrays of bytes will be represented in the following form:

\[ a_0 a_1 a_2 ... a_{15} \]

The bytes and the bit ordering within bytes are derived from the 128-bit input sequence

\[
    \text{input}_0 \ \text{input}_1 \ \text{input}_2 \ ... \ \text{input}_{126} \ \text{input}_{127}
\]

as follows:
\[ a_0 = \{ \text{input}_0, \text{input}_1, \ldots, \text{input}_7 \}; \]
\[ a_1 = \{ \text{input}_8, \text{input}_9, \ldots, \text{input}_{15} \}; \]
\[ \vdots \]
\[ a_{15} = \{ \text{input}_{120}, \text{input}_{121}, \ldots, \text{input}_{127} \}. \]

The pattern can be extended to longer sequences (i.e., for 192- and 256-bit keys), so that, in general,
\[ a_n = \{ \text{input}_{8n}, \text{input}_{8n+1}, \ldots, \text{input}_{8n+7} \}. \] (3.2)

Taking Sections 3.2 and 3.3 together, Fig. 2 shows how bits within each byte are numbered.

| Input bit sequence | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | ... |
|--------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|...|
| Byte number        | 0 |   |   |   |   |   |   |   | 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Bit numbers in byte| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ... |

**Figure 2. Indices for Bytes and Bits.**

### 3.4 The State

Internally, the AES algorithm’s operations are performed on a two-dimensional array of bytes called the **State**. The State consists of four rows of bytes, each containing \( N_b \) bytes, where \( N_b \) is the block length divided by 32. In the State array denoted by the symbol \( s \), each individual byte has two indices, with its row number \( r \) in the range \( 0 \leq r < 4 \) and its column number \( c \) in the range \( 0 \leq c < N_b \). This allows an individual byte of the State to be referred to as either \( s_{r,c} \) or \( s[r,c] \). For this standard, \( N_b = 4 \), i.e., \( 0 \leq c < 4 \) (also see Sec. 6.3).

At the start of the Cipher and Inverse Cipher described in Sec. 5, the input – the array of bytes \( \text{in}_0, \text{in}_1, \ldots, \text{in}_{15} \) – is copied into the State array as illustrated in Fig. 3. The Cipher or Inverse Cipher operations are then conducted on this State array, after which its final value is copied to the output – the array of bytes \( \text{out}_0, \text{out}_1, \ldots, \text{out}_{15} \).

Hence, at the beginning of the Cipher or Inverse Cipher, the input array, \( \text{in} \), is copied to the State array according to the scheme:
\[ s[r, c] = \text{in}[r + 4c] \quad \text{for } 0 \leq r < 4 \text{ and } 0 \leq c < N_b, \] (3.3)
and at the end of the Cipher and Inverse Cipher, the State is copied to the output array $out$ as follows:

$$out[r + 4c] = s[r, c] \quad \text{for } 0 \leq r < 4 \text{ and } 0 \leq c < \text{Nb}. \quad (3.4)$$

### 3.5 The State as an Array of Columns

The four bytes in each column of the State array form 32-bit words, where the row number $r$ provides an index for the four bytes within each word. The state can hence be interpreted as a one-dimensional array of 32 bit words (columns), $w_0...w_3$, where the column number $c$ provides an index into this array. Hence, for the example in Fig. 3, the State can be considered as an array of four words, as follows:

$$w_0 = s_{0,0} \ s_{1,0} \ s_{2,0} \ s_{3,0} \quad w_2 = s_{0,2} \ s_{1,2} \ s_{2,2} \ s_{3,2}$$
$$w_1 = s_{0,1} \ s_{1,1} \ s_{2,1} \ s_{3,1} \quad w_3 = s_{0,3} \ s_{1,3} \ s_{2,3} \ s_{3,3} \quad (3.5)$$

### 4. Mathematical Preliminaries

All bytes in the AES algorithm are interpreted as finite field elements using the notation introduced in Sec. 3.2. Finite field elements can be added and multiplied, but these operations are different from those used for numbers. The following subsections introduce the basic mathematical concepts needed for Sec. 5.

#### 4.1 Addition

The addition of two elements in a finite field is achieved by “adding” the coefficients for the corresponding powers in the polynomials for the two elements. The addition is performed with the XOR operation (denoted by $\oplus$) - i.e., modulo 2 - so that $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, and $0 \oplus 0 = 0$. Consequently, subtraction of polynomials is identical to addition of polynomials.

Alternatively, addition of finite field elements can be described as the modulo 2 addition of corresponding bits in the byte. For two bytes \{a_7a_6a_5a_4a_3a_2a_1a_0\} and \{b_7b_6b_5b_4b_3b_2b_1b_0\}, the sum is \{c_7c_6c_5c_4c_3c_2c_1c_0\}, where each $c_i = a_i \oplus b_i$ (i.e., $c_7 = a_7 \oplus b_7$, $c_6 = a_6 \oplus b_6$, ...$c_0 = a_0 \oplus b_0$).

For example, the following expressions are equivalent to one another:

\[
\begin{align*}
(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) &= x^7 + x^6 + x^4 + x^2 \\
\{01010111\} \oplus \{10000011\} &= \{11010100\} \\
\{57\} \oplus \{83\} &= \{d4\}
\end{align*}
\]

(binary notation); (polynomial notation); (hexadecimal notation).

#### 4.2 Multiplication

In the polynomial representation, multiplication in $\mathbb{GF}(2^8)$ (denoted by $\bullet$) corresponds with the multiplication of polynomials modulo an irreducible polynomial of degree 8. A polynomial is irreducible if its only divisors are one and itself. For the AES algorithm, this irreducible polynomial is

$$m(x) = x^8 + x^4 + x^3 + x + 1, \quad (4.1)$$
or \{01\} \{1b\} in hexadecimal notation.

For example, \{(57) \cdot \{83\} = \{c1\}\}, because

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

and

\[
x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \mod \left(x^8 + x^4 + x^3 + x + 1\right)
\]
\[
= x^7 + x^6 + 1.
\]

The modular reduction by \(m(x)\) ensures that the result will be a binary polynomial of degree less than 8, and thus can be represented by a byte. Unlike addition, there is no simple operation at the byte level that corresponds to this multiplication.

The multiplication defined above is associative, and the element \{01\} is the multiplicative identity. For any non-zero binary polynomial \(b(x)\) of degree less than 8, the multiplicative inverse of \(b(x)\), denoted \(b^{-1}(x)\), can be found as follows: the extended Euclidean algorithm \([7]\) is used to compute polynomials \(a(x)\) and \(c(x)\) such that

\[
b(x)a(x) + m(x)c(x) = 1. \tag{4.2}
\]

Hence, \(a(x) \cdot b(x) \mod m(x) = 1\), which means

\[
b^{-1}(x) = a(x) \mod m(x). \tag{4.3}
\]

Moreover, for any \(a(x)\), \(b(x)\) and \(c(x)\) in the field, it holds that

\[
a(x) \cdot (b(x) + c(x)) = a(x) \cdot b(x) + a(x) \cdot c(x).
\]

It follows that the set of 256 possible byte values, with XOR used as addition and the multiplication defined as above, has the structure of the finite field \(\operatorname{GF}(2^8)\).

### 4.2.1 Multiplication by \(x\)

Multiplying the binary polynomial defined in equation (3.1) with the polynomial \(x\) results in

\[
b_7 x^8 + b_6 x^7 + b_5 x^6 + b_4 x^5 + b_3 x^4 + b_2 x^3 + b_1 x^2 + b_0 x. \tag{4.4}
\]

The result \(x \cdot b(x)\) is obtained by reducing the above result modulo \(m(x)\), as defined in equation (4.1). If \(b_7 = 0\), the result is already in reduced form. If \(b_7 = 1\), the reduction is accomplished by subtracting (i.e., XORing) the polynomial \(m(x)\). It follows that multiplication by \(x\) (i.e., \{000000010\} or \{02\}) can be implemented at the byte level as a left shift and a subsequent conditional bitwise XOR with \{1b\}. This operation on bytes is denoted by \(\times\operatorname{time}()\). Multiplication by higher powers of \(x\) can be implemented by repeated application of \(\times\operatorname{time}()\). By adding intermediate results, multiplication by any constant can be implemented.

For example, \{(57) \cdot \{13\} = \{fe\}\} because
\{57\} \cdot \{02\} = \text{xtime}(\{57\}) = \{ae\} \\
\{57\} \cdot \{04\} = \text{xtime}(\{ae\}) = \{47\} \\
\{57\} \cdot \{08\} = \text{xtime}(\{47\}) = \{8e\} \\
\{57\} \cdot \{10\} = \text{xtime}(\{8e\}) = \{07\},

\text{thus,}

\{57\} \cdot \{13\} = \{57\} \cdot (\{01\} \oplus \{02\} \oplus \{10\}) \\
= \{57\} \oplus \{ae\} \oplus \{07\} \\
= \{fe\}.

### 4.3 Polynomials with Coefficients in GF($2^8$)

Four-term polynomials can be defined - with coefficients that are finite field elements - as:

\[ a(x) = a_3 x^3 + a_2 x^2 + a_1 x + a_0 \]  

(4.5)

which will be denoted as a word in the form \([a_0, a_1, a_2, a_3]\). Note that the polynomials in this section behave somewhat differently than the polynomials used in the definition of finite field elements, even though both types of polynomials use the same indeterminate, \(x\). The coefficients in this section are themselves finite field elements, i.e., bytes, instead of bits; also, the multiplication of four-term polynomials uses a different reduction polynomial, defined below. The distinction should always be clear from the context.

To illustrate the addition and multiplication operations, let

\[ b(x) = b_3 x^3 + b_2 x^2 + b_1 x + b_0 \]  

(4.6)

define a second four-term polynomial. Addition is performed by adding the finite field coefficients of like powers of \(x\). This addition corresponds to an XOR operation between the corresponding bytes in each of the words – in other words, the XOR of the complete word values.

Thus, using the equations of (4.5) and (4.6),

\[ a(x) + b(x) = (a_3 \oplus b_3) x^3 + (a_2 \oplus b_2) x^2 + (a_1 \oplus b_1) x + (a_0 \oplus b_0) \]  

(4.7)

Multiplication is achieved in two steps. In the first step, the polynomial product \(c(x) = a(x) \cdot b(x)\) is algebraically expanded, and like powers are collected to give

\[ c(x) = c_6 x^6 + c_5 x^5 + c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0 \]  

(4.8)

where

\[
\begin{align*}
c_0 &= a_0 \cdot b_0 \\
c_1 &= a_1 \cdot b_0 \oplus a_0 \cdot b_1 \\
c_2 &= a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \\
c_3 &= a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3 \\
c_4 &= a_3 \cdot b_1 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \\
c_5 &= a_3 \cdot b_2 \oplus a_2 \cdot b_3 \\
c_6 &= a_3 \cdot b_3
\end{align*}
\]  

(4.9)
\[ c_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3. \]

The result, \( c(x) \), does not represent a four-byte word. Therefore, the second step of the multiplication is to reduce \( c(x) \) modulo a polynomial of degree 4; the result can be reduced to a polynomial of degree less than 4. For the AES algorithm, this is accomplished with the polynomial \( x^4 + 1 \), so that

\[ x' \mod(x^4 + 1) = x^{i \mod 4}. \]

The modular product of \( a(x) \) and \( b(x) \), denoted by \( a(x) \otimes b(x) \), is given by the four-term polynomial \( d(x) \), defined as follows:

\[
d(x) = d_3 x^3 + d_2 x^2 + d_1 x + d_0
\]

with

\[
\begin{align*}
d_0 &= (a_0 \cdot b_0) \oplus (a_3 \cdot b_1) \oplus (a_2 \cdot b_2) \oplus (a_1 \cdot b_3) \\
d_1 &= (a_1 \cdot b_0) \oplus (a_0 \cdot b_1) \oplus (a_3 \cdot b_2) \oplus (a_2 \cdot b_3) \\
d_2 &= (a_2 \cdot b_0) \oplus (a_1 \cdot b_1) \oplus (a_0 \cdot b_2) \oplus (a_3 \cdot b_3) \\
d_3 &= (a_3 \cdot b_0) \oplus (a_2 \cdot b_1) \oplus (a_1 \cdot b_2) \oplus (a_0 \cdot b_3)
\end{align*}
\]

When \( a(x) \) is a fixed polynomial, the operation defined in equation (4.11) can be written in matrix form as:

\[
\begin{bmatrix}
    d_0 \\
    d_1 \\
    d_2 \\
    d_3
\end{bmatrix} =
\begin{bmatrix}
    a_0 & a_3 & a_2 & a_1 \\
    a_1 & a_0 & a_3 & a_2 \\
    a_2 & a_1 & a_0 & a_3 \\
    a_3 & a_2 & a_1 & a_0
\end{bmatrix}
\begin{bmatrix}
    b_0 \\
    b_1 \\
    b_2 \\
    b_3
\end{bmatrix}
\]

Because \( x^4 + 1 \) is not an irreducible polynomial over GF(2^8), multiplication by a fixed four-term polynomial is not necessarily invertible. However, the AES algorithm specifies a fixed four-term polynomial that does have an inverse (see Sec. 5.1.3 and Sec. 5.3.3):

\[
a(x) = \{03\} x^3 + \{01\} x^2 + \{01\} x + \{02\} \quad (4.14)
\]

\[
a^{-1}(x) = \{0b\} x^3 + \{0d\} x^2 + \{09\} x + \{0e\}. \quad (4.15)
\]

Another polynomial used in the AES algorithm (see the RotWord() function in Sec. 5.2) has \( a_0 = a_1 = a_2 = \{00\} \) and \( a_3 = \{01\} \), which is the polynomial \( x^3 \). Inspection of equation (4.13) above will show that its effect is to form the output word by rotating bytes in the input word. This means that \([b_0, b_1, b_2, b_3]\) is transformed into \([b_1, b_2, b_3, b_0]\).

### 5. Algorithm Specification

For the AES algorithm, the length of the input block, the output block and the State is 128 bits. This is represented by \( Nb = 4 \), which reflects the number of 32-bit words (number of columns) in the State.
For the AES algorithm, the length of the Cipher Key, $K$, is 128, 192, or 256 bits. The key length is represented by $N_k = 4, 6, \text{or } 8$, which reflects the number of 32-bit words (number of columns) in the Cipher Key.

For the AES algorithm, the number of rounds to be performed during the execution of the algorithm is dependent on the key size. The number of rounds is represented by $N_r$, where $N_r = 10$ when $N_k = 4$, $N_r = 12$ when $N_k = 6$, and $N_r = 14$ when $N_k = 8$.

The only Key-Block-Round combinations that conform to this standard are given in Fig. 4. For implementation issues relating to the key length, block size and number of rounds, see Sec. 6.3.

<table>
<thead>
<tr>
<th>Key Length (N_k words)</th>
<th>Block Size (N_b words)</th>
<th>Number of Rounds (N_r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>AES-192</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>AES-256</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 4. Key-Block-Round Combinations.

For both its Cipher and Inverse Cipher, the AES algorithm uses a round function that is composed of four different byte-oriented transformations: 1) byte substitution using a substitution table (S-box), 2) shifting rows of the State array by different offsets, 3) mixing the data within each column of the State array, and 4) adding a Round Key to the State. These transformations (and their inverses) are described in Sec. 5.1.1-5.1.4 and 5.3.1-5.3.4.

The Cipher and Inverse Cipher are described in Sec. 5.1 and Sec. 5.3, respectively, while the Key Schedule is described in Sec. 5.2.

### 5.1 Cipher

At the start of the Cipher, the input is copied to the State array using the conventions described in Sec. 3.4. After an initial Round Key addition, the State array is transformed by implementing a round function 10, 12, or 14 times (depending on the key length), with the final round differing slightly from the first $N_r - 1$ rounds. The final State is then copied to the output as described in Sec. 3.4.

The round function is parameterized using a key schedule that consists of a one-dimensional array of four-byte words derived using the Key Expansion routine described in Sec. 5.2.

The Cipher is described in the pseudo code in Fig. 5. The individual transformations - `SubBytes()`, `ShiftRows()`, `MixColumns()`, and `AddRoundKey()` – process the State and are described in the following subsections. In Fig. 5, the array $w[]$ contains the key schedule, which is described in Sec. 5.2.

As shown in Fig. 5, all $N_r$ rounds are identical with the exception of the final round, which does not include the `MixColumns()` transformation.
Appendix B presents an example of the Cipher, showing values for the State array at the beginning of each round and after the application of each of the four transformations described in the following sections.

```plaintext
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in

    AddRoundKey(state, w[0, Nb-1])  // See Sec. 5.1.4

    for round = 1 step 1 to Nr-1
        SubBytes(state)  // See Sec. 5.1.1
        ShiftRows(state)  // See Sec. 5.1.2
        MixColumns(state)  // See Sec. 5.1.3
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for

    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

    out = state
end
```

Figure 5. Pseudo Code for the Cipher.

5.1.1 SubBytes() Transformation

The SubBytes() transformation is a non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-box). This S-box (Fig. 7), which is invertible, is constructed by composing two transformations:

1. Take the multiplicative inverse in the finite field GF(2^8), described in Sec. 4.2; the element \{00\} is mapped to itself.

2. Apply the following affine transformation (over GF(2)):

   \[ b'_i = b_i \oplus b_{(i+4)\text{mod 8}} \oplus b_{(i+5)\text{mod 8}} \oplus b_{(i+6)\text{mod 8}} \oplus b_{(i+7)\text{mod 8}} \oplus c_i \]  \hspace{1cm} (5.1)

for \(0 \leq i < 8\), where \(b_i\) is the \(i^{th}\) bit of the byte, and \(c_i\) is the \(i^{th}\) bit of a byte \(c\) with the value \{63\} or \{01100011\}. Here and elsewhere, a prime on a variable (e.g., \(b'_i\)) indicates that the variable is to be updated with the value on the right.

In matrix form, the affine transformation element of the S-box can be expressed as:

---

1 The various transformations (e.g., SubBytes(), ShiftRows(), etc.) act upon the State array that is addressed by the 'state' pointer. AddRoundKey() uses an additional pointer to address the Round Key.
Figure 6 illustrates the effect of the \textbf{SubBytes()} transformation on the State.

\begin{equation}
\begin{bmatrix}
B_0 \\
B_1 \\
B_2 \\
B_3 \\
B_4 \\
B_5 \\
B_6 \\
B_7
\end{bmatrix} = \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 & 1 & 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{bmatrix}
1 \\
1 \\
0 \\
0 \\
0 \\
1 \\
1 \\
0
\end{bmatrix}.
\end{equation}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{s-box.png}
\caption{\textbf{SubBytes()} applies the S-box to each byte of the State.}
\end{figure}

The S-box used in the \textbf{SubBytes()} transformation is presented in hexadecimal form in Fig. 7. For example, if \(s_{1,1} = \{53\}\), then the substitution value would be determined by the intersection of the row with index ‘5’ and the column with index ‘3’ in Fig. 7. This would result in \(s'_{1,1}\) having a value of \{ed\}.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline
\textbf{x} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \textbf{a} & b & c & d & e & f \\
\hline
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 & 0d & ef & aa & fb & 43 & 4d & 33 & 85 & 45 & f9 & 02 & 7f & 0e & 27 & b2 & 76 \\
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 & ld & 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 \\
\hline
\end{tabular}
\caption{S-box: substitution values for the byte xy (in hexadecimal format).}
\end{table}
5.1.2 ShiftRows() Transformation

In the ShiftRows() transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, \( r = 0 \), is not shifted.

Specifically, the ShiftRows() transformation proceeds as follows:

\[
s_{r,c}' = s_{r,(c+shift(r,Nb)) \mod Nb} \quad \text{for} \quad 0 < r < 4 \quad \text{and} \quad 0 \leq c < Nb,
\]

where the shift value \( shift(r,Nb) \) depends on the row number, \( r \), as follows (recall that \( Nb = 4 \)):

\[
shift(1,4) = 1; \quad shift(2,4) = 2; \quad shift(3,4) = 3.
\]

This has the effect of moving bytes to “lower” positions in the row (i.e., lower values of \( c \) in a given row), while the “lowest” bytes wrap around into the “top” of the row (i.e., higher values of \( c \) in a given row).

Figure 8 illustrates the ShiftRows() transformation.

![ShiftRows Diagram](image)

**Figure 8.** ShiftRows() cyclically shifts the last three rows in the State.

5.1.3 MixColumns() Transformation

The MixColumns() transformation operates on the State column-by-column, treating each column as a four-term polynomial as described in Sec. 4.3. The columns are considered as polynomials over GF\( (2^8) \) and multiplied modulo \( x^4 + 1 \) with a fixed polynomial \( a(x) \), given by

\[
a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}.
\]

As described in Sec. 4.3, this can be written as a matrix multiplication. Let

\[
s'(x) = a(x) \otimes s(x)
\]
As a result of this multiplication, the four bytes in a column are replaced by the following:

\[
\begin{align*}
{s'_{0,c}} &= ([02] \cdot s_{0,c}) \oplus ([03] \cdot s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\
{s'_{1,c}} &= s_{0,c} \oplus ([02] \cdot s_{1,c}) \oplus ([03] \cdot s_{2,c}) \oplus s_{3,c} \\
{s'_{2,c}} &= s_{0,c} \oplus s_{1,c} \oplus ([02] \cdot s_{2,c}) \oplus ([03] \cdot s_{3,c}) \\
{s'_{3,c}} &= ([03] \cdot s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus ([02] \cdot s_{3,c}).
\end{align*}
\]

Figure 9 illustrates the \textbf{MixColumns()} transformation.

5.1.4 \textbf{AddRoundKey()} Transformation

In the \textbf{AddRoundKey()} transformation, a Round Key is added to the State by a simple bitwise XOR operation. Each Round Key consists of \(Nb\) words from the key schedule (described in Sec. 5.2). Those \(Nb\) words are each added into the columns of the State, such that

\[
[w_{\text{round} \cdot Nb + c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \quad \text{for } 0 \leq c < Nb,
\]

where \([w_i]\) are the key schedule words described in Sec. 5.2, and \(\text{round}\) is a value in the range \(0 \leq \text{round} \leq Nr\). In the Cipher, the initial Round Key addition occurs when \(\text{round} = 0\), prior to the first application of the round function (see Fig. 5). The application of the \textbf{AddRoundKey()} transformation to the \(Nr\) rounds of the Cipher occurs when \(1 \leq \text{round} \leq Nr\).

The action of this transformation is illustrated in Fig. 10, where \(l = \text{round} \cdot Nb\). The byte address within words of the key schedule was described in Sec. 3.1.
5.2 Key Expansion

The AES algorithm takes the Cipher Key, $K$, and performs a Key Expansion routine to generate a key schedule. The Key Expansion generates a total of $Nb \times (Nr + 1)$ words: the algorithm requires an initial set of $Nb$ words, and each of the $Nr$ rounds requires $Nb$ words of key data. The resulting key schedule consists of a linear array of 4-byte words, denoted $[w_i]$, with $i$ in the range $0 \leq i < Nb(Nr + 1)$.

The expansion of the input key into the key schedule proceeds according to the pseudo code in Fig. 11.

$\text{SubWord()}$ is a function that takes a four-byte input word and applies the S-box (Sec. 5.1.1, Fig. 7) to each of the four bytes to produce an output word. The function $\text{RotWord()}$ takes a word $[a_0,a_1,a_2,a_3]$ as input, performs a cyclic permutation, and returns the word $[a_1,a_2,a_3,a_0]$. The round constant word array, $\text{Rcon}[i]$, contains the values given by $[x^{i-1},\{00\},\{00\},\{00\}]$, with $x^{i-1}$ being powers of $x$ ($x$ is denoted as $\{02\}$) in the field GF(2$^8$), as discussed in Sec. 4.2 (note that $i$ starts at 1, not 0).

From Fig. 11, it can be seen that the first $Nk$ words of the expanded key are filled with the Cipher Key. Every following word, $w[i]$, is equal to the XOR of the previous word, $w[i-1]$, and the word $Nk$ positions earlier, $w[i-Nk]$. For words in positions that are a multiple of $Nk$, a transformation is applied to $w[i-1]$ prior to the XOR, followed by an XOR with a round constant, $\text{Rcon}[i]$. This transformation consists of a cyclic shift of the bytes in a word ($\text{RotWord()}$), followed by the application of a table lookup to all four bytes of the word ($\text{SubWord()}$).

It is important to note that the Key Expansion routine for 256-bit Cipher Keys ($Nk = 8$) is slightly different than for 128- and 192-bit Cipher Keys. If $Nk = 8$ and $i-4$ is a multiple of $Nk$, then $\text{SubWord()}$ is applied to $w[i-1]$ prior to the XOR.

Figure 10. $\text{AddRoundKey()}$ XORs each column of the State with a word from the key schedule.
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
    word temp
    i = 0
    while (i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i+1
    end while
    i = Nk
    while (i < Nb * (Nr+1)]
        temp = w[i-1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end

Note that Nk=4, 6, and 8 do not all have to be implemented; they are all included in the conditional statement above for conciseness. Specific implementation requirements for the Cipher Key are presented in Sec. 6.1.

Figure 11. Pseudo Code for Key Expansion. ²

Appendix A presents examples of the Key Expansion.

5.3 Inverse Cipher

The Cipher transformations in Sec. 5.1 can be inverted and then implemented in reverse order to produce a straightforward Inverse Cipher for the AES algorithm. The individual transformations used in the Inverse Cipher - InvShiftRows(), InvSubBytes(), InvMixColumns(), and AddRoundKey() – process the State and are described in the following subsections.

The Inverse Cipher is described in the pseudo code in Fig. 12. In Fig. 12, the array w[] contains the key schedule, which was described previously in Sec. 5.2.

² The functions SubWord() and RotWord() return a result that is a transformation of the function input, whereas the transformations in the Cipher and Inverse Cipher (e.g., ShiftRows(), SubBytes(), etc.) transform the State array that is addressed by the ‘state’ pointer.
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) // See Sec. 5.1.4
    for round = Nr-1 step -1 downto 1
        InvShiftRows(state) // See Sec. 5.3.1
        InvSubBytes(state) // See Sec. 5.3.2
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
        InvMixColumns(state) // See Sec. 5.3.3
    end for
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[0, Nb-1])
    out = state
end

Figure 12. Pseudo Code for the Inverse Cipher.³

5.3.1 InvShiftRows() Transformation
InvShiftRows() is the inverse of the ShiftRows() transformation. The bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, \( r = 0 \), is not shifted. The bottom three rows are cyclically shifted by \( Nb - shift(r,Nb) \) bytes, where the shift value \( shift(r,Nb) \) depends on the row number, and is given in equation (5.4) (see Sec. 5.1.2).

Specifically, the InvShiftRows() transformation proceeds as follows:

\[
    s'_{r,c+shift(r,Nb)) mod Nb} = s_{r,c} \quad \text{for } 0 < r < 4 \quad \text{and} \quad 0 \leq c < Nb
\]  
(5.8)

Figure 13 illustrates the InvShiftRows() transformation.

³ The various transformations (e.g., InvSubBytes(), InvShiftRows(), etc.) act upon the State array that is addressed by the state pointer. AddRoundKey() uses an additional pointer to address the Round Key.
5.3.2 \textit{InvSubBytes()} Transformation

\textit{InvSubBytes()} is the inverse of the byte substitution transformation, in which the inverse S-box is applied to each byte of the State. This is obtained by applying the inverse of the affine transformation (5.1) followed by taking the multiplicative inverse in GF(2$^8$).

The inverse S-box used in the \textit{InvSubBytes()} transformation is presented in Fig. 14:

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{inverse_s_box.png}
\caption{Inverse S-box: substitution values for the byte $xy$ (in hexadecimal format).}
\end{figure}
5.3.3 InvMixColumns() Transformation

InvMixColumns() is the inverse of the MixColumns() transformation. InvMixColumns() operates on the State column-by-column, treating each column as a four-term polynomial as described in Sec. 4.3. The columns are considered as polynomials over GF(2^8) and multiplied modulo \( x^4 + 1 \) with a fixed polynomial \( a^{-1}(x) \), given by

\[
a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}.
\]  

As described in Sec. 4.3, this can be written as a matrix multiplication. Let

\[
\begin{bmatrix}
s_0, c \\
s_1, c \\
s_2, c \\
s_3, c \\
\end{bmatrix} = \begin{bmatrix}
0e & 0b & 0d & 09 \\
09 & 0e & 0b & 0d \\
0d & 09 & 0e & 0b \\
0b & 0d & 09 & 0e \\
\end{bmatrix} \begin{bmatrix}
s_0, c \\
s_1, c \\
s_2, c \\
s_3, c \\
\end{bmatrix}
\]  

for \( 0 \leq c < Nb \). (5.10)

As a result of this multiplication, the four bytes in a column are replaced by the following:

\[
s'_{0,c} = (\{0e\} \cdot s_{0,c} \oplus (\{0b\} \cdot s_{1,c}) \oplus (\{0d\} \cdot s_{2,c}) \oplus (\{09\} \cdot s_{3,c})
\]

\[
s'_{1,c} = (\{09\} \cdot s_{0,c} \oplus (\{0e\} \cdot s_{1,c}) \oplus (\{0b\} \cdot s_{2,c}) \oplus (\{0d\} \cdot s_{3,c})
\]

\[
s'_{2,c} = (\{0d\} \cdot s_{0,c} \oplus (\{09\} \cdot s_{1,c}) \oplus (\{0e\} \cdot s_{2,c}) \oplus (\{0b\} \cdot s_{3,c})
\]

\[
s'_{3,c} = (\{0b\} \cdot s_{0,c} \oplus (\{0d\} \cdot s_{1,c}) \oplus (\{09\} \cdot s_{2,c}) \oplus (\{0e\} \cdot s_{3,c})
\]

5.3.4 Inverse of the AddRoundKey() Transformation

AddRoundKey(), which was described in Sec. 5.1.4, is its own inverse, since it only involves an application of the XOR operation.

5.3.5 Equivalent Inverse Cipher

In the straightforward Inverse Cipher presented in Sec. 5.3 and Fig. 12, the sequence of the transformations differs from that of the Cipher, while the form of the key schedules for encryption and decryption remains the same. However, several properties of the AES algorithm allow for an Equivalent Inverse Cipher that has the same sequence of transformations as the Cipher (with the transformations replaced by their inverses). This is accomplished with a change in the key schedule.

The two properties that allow for this Equivalent Inverse Cipher are as follows:

1. The SubBytes() and ShiftRows() transformations commute; that is, a SubBytes() transformation immediately followed by a ShiftRows() transformation is equivalent to a ShiftRows() transformation immediately followed buy a SubBytes() transformation. The same is true for their inverses, InvSubBytes() and InvShiftRows.
2. The column mixing operations - \texttt{MixColumns()} and \texttt{InvMixColumns()} - are linear with respect to the column input, which means

\[
\text{InvMixColumns(state XOR Round Key)} = \text{InvMixColumns(state)} \oplus \text{InvMixColumns(Round Key)}.
\]

These properties allow the order of \texttt{InvSubBytes()} and \texttt{InvShiftRows()} transformations to be reversed. The order of the \texttt{AddRoundKey()} and \texttt{InvMixColumns()} transformations can also be reversed, provided that the columns (words) of the decryption key schedule are modified using the \texttt{InvMixColumns()} transformation.

The equivalent inverse cipher is defined by reversing the order of the \texttt{InvSubBytes()} and \texttt{InvShiftRows()} transformations shown in Fig. 12, and by reversing the order of the \texttt{AddRoundKey()} and \texttt{InvMixColumns()} transformations used in the “round loop” after first modifying the decryption key schedule for \texttt{round} = 1 to \texttt{Nr}-1 using the \texttt{InvMixColumns()} transformation. The first and last \texttt{Nb} words of the decryption key schedule shall not be modified in this manner.

Given these changes, the resulting Equivalent Inverse Cipher offers a more efficient structure than the Inverse Cipher described in Sec. 5.3 and Fig. 12. Pseudo code for the Equivalent Inverse Cipher appears in Fig. 15. (The word array \texttt{dw[]} contains the modified decryption key schedule. The modification to the Key Expansion routine is also provided in Fig. 15.)
EqInvCipher(byte in[4*Nb], byte out[4*Nb], word dw[Nb*(Nr+1)])
begin
  byte state[4,Nb]

  state = in

  AddRoundKey(state, dw[Nr*Nb, (Nr+1)*Nb-1])

  for round = Nr-1 step -1 downto 1
    InvSubBytes(state)
    InvShiftRows(state)
    InvMixColumns(state)
    AddRoundKey(state, dw[round*Nb, (round+1)*Nb-1])
  end for

  InvSubBytes(state)
  InvShiftRows(state)
  AddRoundKey(state, dw[0, Nb-1])

  out = state
end

For the Equivalent Inverse Cipher, the following pseudo code is added at the end of the Key Expansion routine (Sec. 5.2):

```plaintext
for i = 0 step 1 to (Nr+1)*Nb-1
  dw[i] = w[i]
end for

for round = 1 step 1 to Nr-1
  InvMixColumns(dw[round*Nb, (round+1)*Nb-1])  // note change of type
end for
```

Note that, since InvMixColumns operates on a two-dimensional array of bytes while the Round Keys are held in an array of words, the call to InvMixColumns in this code sequence involves a change of type (i.e. the input to InvMixColumns() is normally the State array, which is considered to be a two-dimensional array of bytes, whereas the input here is a Round Key computed as a one-dimensional array of words).

Figure 15. Pseudo Code for the Equivalent Inverse Cipher.

6. Implementation Issues

6.1 Key Length Requirements

An implementation of the AES algorithm shall support at least one of the three key lengths specified in Sec. 5: 128, 192, or 256 bits (i.e., \(N_k = 4, 6, \) or 8, respectively). Implementations
may optionally support two or three key lengths, which may promote the interoperability of algorithm implementations.

### 6.2 Keying Restrictions

No weak or semi-weak keys have been identified for the AES algorithm, and there is no restriction on key selection.

### 6.3 Parameterization of Key Length, Block Size, and Round Number

This standard explicitly defines the allowed values for the key length ($N_k$), block size ($N_b$), and number of rounds ($N_r$) – see Fig. 4. However, future reaffirmations of this standard could include changes or additions to the allowed values for those parameters. Therefore, implementers may choose to design their AES implementations with future flexibility in mind.

### 6.4 Implementation Suggestions Regarding Various Platforms

Implementation variations are possible that may, in many cases, offer performance or other advantages. Given the same input key and data (plaintext or ciphertext), any implementation that produces the same output (ciphertext or plaintext) as the algorithm specified in this standard is an acceptable implementation of the AES.

Reference [3] and other papers located at Ref. [1] include suggestions on how to efficiently implement the AES algorithm on a variety of platforms.
Appendix A - Key Expansion Examples

This appendix shows the development of the key schedule for various key sizes. Note that multi-byte values are presented using the notation described in Sec. 3. The intermediate values produced during the development of the key schedule (see Sec. 5.2) are given in the following table (all values are in hexadecimal format, with the exception of the index column (i)).

### A.1 Expansion of a 128-bit Cipher Key

This section contains the key expansion of the following cipher key:

\[
\text{Cipher Key} = 2b\ 7e\ 15\ 16\ 28\ ae\ d2\ a6\ f7\ 15\ 88\ 09\ cf\ 4f\ 3c
\]

for \( Nk = 4 \), which results in

\[
\begin{align*}
w_0 &= 2b7e1516 \\
w_1 &= 28aed2a6 \\
w_2 &= abf71588 \\
w_3 &= 09cf4f3c
\end{align*}
\]

<table>
<thead>
<tr>
<th>i (dec)</th>
<th>temp</th>
<th>After RotWord()</th>
<th>After SubWord()</th>
<th>Rcon[i/Nk]</th>
<th>After XOR with Rcon</th>
<th>w[i-Nk]</th>
<th>w[i]= temp XOR w[i-Nk]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>09cf4f3c</td>
<td>cf4f3c09</td>
<td>8a84eb01</td>
<td>01000000</td>
<td>8b84eb01</td>
<td>2b7e1516</td>
<td>a0fafe17</td>
</tr>
<tr>
<td>5</td>
<td>a0fafe17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>88542cb1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>23a33939</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2a6c7605</td>
<td>6c76052a</td>
<td>50386be5</td>
<td>02000000</td>
<td>52386be5</td>
<td>a0fafe17</td>
<td>f2c295f2</td>
</tr>
<tr>
<td>9</td>
<td>f2c295f2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>7a96b943</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>5935807a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>7359f67f</td>
<td>59f67f73</td>
<td>cb42d28f</td>
<td>04000000</td>
<td>cf42d28f</td>
<td>f2c295f2</td>
<td>3d80477d</td>
</tr>
<tr>
<td>13</td>
<td>3d80477d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>4716fe3e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1e237e44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>6d7a883b</td>
<td>7a883b6d</td>
<td>dac4e23c</td>
<td>08000000</td>
<td>d2c4e23c</td>
<td>3d80477d</td>
<td>ef44a541</td>
</tr>
<tr>
<td>17</td>
<td>ef44a541</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>a8525b7f</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>b671253b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>db0bad00</td>
<td>0bad00db</td>
<td>2b9563b9</td>
<td>10000000</td>
<td>3b9563b9</td>
<td>ef44a541</td>
<td>d4d1c6f8</td>
</tr>
<tr>
<td>21</td>
<td>d4d1c6f8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>7c839d87</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>caf2b8bc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A.2 Expansion of a 192-bit Cipher Key

This section contains the key expansion of the following cipher key:

\[
\text{Cipher Key} = \begin{array}{cccccccccccccccc}
8e & 73 & b0 & f7 & da & 0e & 64 & 52 & c8 & 10 & f3 & 2b \\
& 80 & 90 & 79 & e5 & 62 & f8 & ea & d2 & 52 & 2c & 6b & 7b
\end{array}
\]

for \( Nk = 6 \), which results in

\[
\begin{align*}
\text{w}_0 &= 8e73b0f7 \\
\text{w}_1 &= da0e6452 \\
\text{w}_2 &= c810f32b \\
\text{w}_3 &= 809079e5 \\
\text{w}_4 &= 62f8ead2 \\
\text{w}_5 &= 522c6b7b
\end{align*}
\]
### A.3 Expansion of a 256-bit Cipher Key

This section contains the key expansion of the following cipher key:

\[
\text{Cipher Key} = \begin{array}{cccccccccccccccc}
60 & 3d & eb & 10 & 15 & ca & 71 & be & 2b & 73 & ae & f0 & 85 & 7d & 77 & 81 \\
1f & 35 & 2c & 07 & 3b & 61 & 08 & d7 & 2d & 98 & 10 & a3 & 09 & 14 & df & f4
\end{array}
\]

for \( Nk = 8 \), which results in

\[
\begin{align*}
\text{w}_0 &= 603deb10 \\
\text{w}_1 &= 15ca71be \\
\text{w}_2 &= 2b73ae0f \\
\text{w}_3 &= 857d7781 \\
\text{w}_4 &= 1f352c07 \\
\text{w}_5 &= 3b6108d7 \\
\text{w}_6 &= 2d9810a3 \\
\text{w}_7 &= 0914dff4
\end{align*}
\]

<table>
<thead>
<tr>
<th>( i ) (dec)</th>
<th>temp</th>
<th>After RotWord()</th>
<th>After SubWord()</th>
<th>Rcon[i/Nk]</th>
<th>After XOR with Rcon</th>
<th>w[i-Nk]</th>
<th>w[i] = temp XOR w[i-Nk]</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0914dff4</td>
<td>14dff409</td>
<td>01000000</td>
<td>fb9ebf01</td>
<td>603deb10</td>
<td>9ba35411</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9ba35411</td>
<td></td>
<td>15ca71be</td>
<td>8e6925af</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>8e6925af</td>
<td></td>
<td>2b73ae0f</td>
<td>a51a8b5f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>a51a8b5f</td>
<td></td>
<td>857d7781</td>
<td>2067fcde</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2067fcde</td>
<td>b785b01d</td>
<td>1f352c07</td>
<td>a8b09c1a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>a8b09c1a</td>
<td></td>
<td>3b6108d7</td>
<td>93d194cd</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>93d194cd</td>
<td></td>
<td>2d9810a3</td>
<td>be49846e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>be49846e</td>
<td></td>
<td>0914dff4</td>
<td>b75d5b9a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>b75d5b9a</td>
<td>5d5b9ab7</td>
<td>4c39b8a9</td>
<td>02000000</td>
<td>9ba35411</td>
<td>d59aecb8</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>d59aecb8</td>
<td></td>
<td>8e6925af</td>
<td>5bf3c917</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>5bf3c917</td>
<td></td>
<td>a51a8b5f</td>
<td>fee94248</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>fee94248</td>
<td></td>
<td>2067fcde</td>
<td>de8ebe96</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>de8ebe96</td>
<td>1d19ae90</td>
<td>a8b09c1a</td>
<td>b5a9328a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>b5a9328a</td>
<td></td>
<td>93d194cd</td>
<td>2678a647</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>2678a647</td>
<td></td>
<td>be49846e</td>
<td>98312229</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

30
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>98312229</td>
<td></td>
<td>b75d5b9a</td>
<td>2f6c79b3</td>
</tr>
<tr>
<td>24</td>
<td>2f6c79b3</td>
<td>6c79b32f</td>
<td>50b66d15</td>
<td>04000000</td>
</tr>
<tr>
<td>25</td>
<td>812c81ad</td>
<td></td>
<td>5bf3c917</td>
<td>dadf48ba</td>
</tr>
<tr>
<td>26</td>
<td>dadf48ba</td>
<td></td>
<td>fee94248</td>
<td>24360af2</td>
</tr>
<tr>
<td>27</td>
<td>24360af2</td>
<td></td>
<td>de8ebe96</td>
<td>fab8b464</td>
</tr>
<tr>
<td>28</td>
<td>fab8b464</td>
<td>2d6c8d43</td>
<td>b5a9328a</td>
<td>98c5bfc9</td>
</tr>
<tr>
<td>29</td>
<td>98c5bfc9</td>
<td></td>
<td>2678a647</td>
<td>bebd198e</td>
</tr>
<tr>
<td>30</td>
<td>bebd198e</td>
<td></td>
<td>98312229</td>
<td>268c3ba7</td>
</tr>
<tr>
<td>31</td>
<td>268c3ba7</td>
<td></td>
<td>2f6c79b3</td>
<td>09e04214</td>
</tr>
<tr>
<td>32</td>
<td>09e04214</td>
<td>e0421409</td>
<td>e12cfa01</td>
<td>08000000</td>
</tr>
<tr>
<td>33</td>
<td>68007bac</td>
<td></td>
<td>dadf48ba</td>
<td>b2df3316</td>
</tr>
<tr>
<td>34</td>
<td>b2df3316</td>
<td></td>
<td>24360af2</td>
<td>96e939e4</td>
</tr>
<tr>
<td>35</td>
<td>96e939e4</td>
<td></td>
<td>fab8b464</td>
<td>6c518d80</td>
</tr>
<tr>
<td>36</td>
<td>6c518d80</td>
<td>50d15dcd</td>
<td>98c5bfc9</td>
<td>c814e204</td>
</tr>
<tr>
<td>37</td>
<td>c814e204</td>
<td></td>
<td>bebd198e</td>
<td>76a9fb8a</td>
</tr>
<tr>
<td>38</td>
<td>76a9fb8a</td>
<td></td>
<td>268c3ba7</td>
<td>5025c02d</td>
</tr>
<tr>
<td>39</td>
<td>5025c02d</td>
<td></td>
<td>09e04214</td>
<td>59c58239</td>
</tr>
<tr>
<td>40</td>
<td>59c58239</td>
<td>c5823959</td>
<td>a61312cb</td>
<td>10000000</td>
</tr>
<tr>
<td>41</td>
<td>de136967</td>
<td></td>
<td>b2df3316</td>
<td>6ccc5a71</td>
</tr>
<tr>
<td>42</td>
<td>6ccc5a71</td>
<td></td>
<td>96e939e4</td>
<td>fa256395</td>
</tr>
<tr>
<td>43</td>
<td>fa256395</td>
<td></td>
<td>6c518d80</td>
<td>9674ee15</td>
</tr>
<tr>
<td>44</td>
<td>9674ee15</td>
<td>90922859</td>
<td>c814e204</td>
<td>5886ca5d</td>
</tr>
<tr>
<td>45</td>
<td>5886ca5d</td>
<td></td>
<td>76a9fb8a</td>
<td>2e2f31d7</td>
</tr>
<tr>
<td>46</td>
<td>2e2f31d7</td>
<td></td>
<td>5025c02d</td>
<td>7e0af1fa</td>
</tr>
<tr>
<td>47</td>
<td>7e0af1fa</td>
<td></td>
<td>59c58239</td>
<td>27cf73c3</td>
</tr>
<tr>
<td>48</td>
<td>27cf73c3</td>
<td>cf73c327</td>
<td>8a8f2ecc</td>
<td>20000000</td>
</tr>
<tr>
<td>49</td>
<td>749c47ab</td>
<td></td>
<td>6ccc5a71</td>
<td>18501dda</td>
</tr>
<tr>
<td>50</td>
<td>18501dda</td>
<td></td>
<td>fa256395</td>
<td>e2757e4f</td>
</tr>
<tr>
<td>51</td>
<td>e2757e4f</td>
<td></td>
<td>9674ee15</td>
<td>7401905a</td>
</tr>
<tr>
<td>52</td>
<td>7401905a</td>
<td>927c60be</td>
<td>5886ca5d</td>
<td>cafaaae3</td>
</tr>
<tr>
<td>53</td>
<td>cafaaae3</td>
<td></td>
<td>2e2f31d7</td>
<td>e4d59b34</td>
</tr>
<tr>
<td>54</td>
<td>e4d59b34</td>
<td></td>
<td>7e0af1fa</td>
<td>9adf6ace</td>
</tr>
<tr>
<td>55</td>
<td>9adf6ace</td>
<td></td>
<td>27cf73c3</td>
<td>bd10190d</td>
</tr>
<tr>
<td>56</td>
<td>bd10190d</td>
<td>10190dbd</td>
<td>cad4d77a</td>
<td>40000000</td>
</tr>
<tr>
<td>57</td>
<td>fe4890d1</td>
<td></td>
<td>18501dda</td>
<td>e6188d0b</td>
</tr>
<tr>
<td>58</td>
<td>e6188d0b</td>
<td></td>
<td>e2757e4f</td>
<td>046df344</td>
</tr>
<tr>
<td>----</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>59</td>
<td>046df344</td>
<td></td>
<td>7401905a</td>
<td>706c631e</td>
</tr>
</tbody>
</table>
## Appendix B – Cipher Example

The following diagram shows the values in the State array as the Cipher progresses for a block length and a Cipher Key length of 16 bytes each (i.e., \( Nb = 4 \) and \( Nk = 4 \)).

\[
\text{Input } = \begin{array}{cccccccccccccccc}
32 & 43 & f6 & a8 & 88 & 5a & 30 & 8d & 31 & 31 & 98 & a2 & e0 & 37 & 07 & 34 \\
\end{array}
\]

\[
\text{Cipher Key } = \begin{array}{cccccccccccccccc}
2b & 7e & 15 & 16 & 28 & ae & d2 & a6 & ab & f7 & 15 & 88 & 09 & cf & 4f & 3c \\
\end{array}
\]

The Round Key values are taken from the Key Expansion example in Appendix A.

<table>
<thead>
<tr>
<th>Round Number</th>
<th>Start of Round</th>
<th>After SubBytes</th>
<th>After ShiftRows</th>
<th>After MixColumns</th>
<th>Round Key Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32 88 31 e0</td>
<td></td>
<td></td>
<td></td>
<td>2b 28 ab 09</td>
</tr>
<tr>
<td></td>
<td>45 5a 31 37</td>
<td></td>
<td></td>
<td></td>
<td>7e ae f7 cf</td>
</tr>
<tr>
<td></td>
<td>f6 30 98 07</td>
<td></td>
<td></td>
<td></td>
<td>15 d2 15 4f</td>
</tr>
<tr>
<td></td>
<td>a8 8d a2 34</td>
<td></td>
<td></td>
<td></td>
<td>16 a6 88 3c</td>
</tr>
<tr>
<td></td>
<td>19 a0 9a e9</td>
<td>d4 e0 b8 1e</td>
<td>d4 e0 b8 1e</td>
<td>04 e0 48 28</td>
<td>a0 88 23 2a</td>
</tr>
<tr>
<td></td>
<td>3d f4 c6 f8</td>
<td>27 bf b4 41</td>
<td>bf b4 41 27</td>
<td>66 cb f8 06</td>
<td>fa 54 a3 6c</td>
</tr>
<tr>
<td></td>
<td>e3 e2 8d 48</td>
<td>11 98 5d 52</td>
<td>5d 52 11 98</td>
<td>81 19 d3 26</td>
<td>fe 2c 39 76</td>
</tr>
<tr>
<td></td>
<td>be 2b 2a 08</td>
<td>ae f1 e5 30</td>
<td>30 ae f1 e5</td>
<td>m5 9a 7a 4c</td>
<td>17 b1 39 05</td>
</tr>
<tr>
<td>2</td>
<td>a4 68 6b 02</td>
<td>49 45 7f 77</td>
<td>49 45 7f 77</td>
<td>58 1b db 1b</td>
<td>f2 7a 59 73</td>
</tr>
<tr>
<td></td>
<td>9c 9f 5b 6a</td>
<td>de db 39 02</td>
<td>db 39 02 de</td>
<td>4d 4b e7 6b</td>
<td>c2 96 35 59</td>
</tr>
<tr>
<td></td>
<td>7f 35 ea 50</td>
<td>d2 96 87 53</td>
<td>87 53 d2 96</td>
<td>ca 5a ca b0</td>
<td>95 b9 80 f6</td>
</tr>
<tr>
<td></td>
<td>f2 2b 43 49</td>
<td>89 f1 1a 3b</td>
<td>3b 89 f1 1a</td>
<td>f1 ac a8 a5</td>
<td>f2 43 7a 7f</td>
</tr>
<tr>
<td>3</td>
<td>aa 61 82 6b</td>
<td>ac ef 13 45</td>
<td>ac ef 13 45</td>
<td>75 20 53 bb</td>
<td>3d 47 1e 6d</td>
</tr>
<tr>
<td></td>
<td>8f dd d2 32</td>
<td>73 c1 b5 23</td>
<td>c1 b5 23 73</td>
<td>ec 0b c0 25</td>
<td>80 16 23 7a</td>
</tr>
<tr>
<td></td>
<td>5f e3 4a 46</td>
<td>cf 11 d6 5a</td>
<td>d6 5a cf 11</td>
<td>09 63 cf d0</td>
<td>47 fe 7e 88</td>
</tr>
<tr>
<td></td>
<td>03 ef d2 9a</td>
<td>7b df b5 b8</td>
<td>b8 7b df b5</td>
<td>93 33 7c dc</td>
<td>7d 3e 44 3b</td>
</tr>
<tr>
<td>4</td>
<td>48 67 4d d6</td>
<td>52 85 e3 f6</td>
<td>52 85 e3 f6</td>
<td>0f 60 6f 5e</td>
<td>ef a8 b6 db</td>
</tr>
<tr>
<td></td>
<td>6c 1d e3 5f</td>
<td>50 a4 11 cf</td>
<td>a4 11 cf 50</td>
<td>d6 31 c0 b3</td>
<td>44 52 71 0b</td>
</tr>
<tr>
<td></td>
<td>4e 9d b1 58</td>
<td>2f 5e c8 6a</td>
<td>c8 6a 2f 5e</td>
<td>da 38 10 13</td>
<td>a5 5b 25 ad</td>
</tr>
<tr>
<td></td>
<td>ee 0d 38 e7</td>
<td>28 d7 07 94</td>
<td>94 28 d7 07</td>
<td>a9 bf 6b 01</td>
<td>41 7f 3b 00</td>
</tr>
<tr>
<td>5</td>
<td>e0 c8 d9 85</td>
<td>e1 e8 35 97</td>
<td>e1 e8 35 97</td>
<td>25 bd b6 4c</td>
<td>d4 7c ca 11</td>
</tr>
<tr>
<td></td>
<td>92 63 b1 b8</td>
<td>4f fb c8 6c</td>
<td>fb c8 6c 4f</td>
<td>d1 11 3a 4c</td>
<td>d4 d1 83 f2 f9</td>
</tr>
<tr>
<td></td>
<td>7f 63 35 be</td>
<td>d2 fb 96 ae</td>
<td>96 ae d2 fb</td>
<td>a9 d1 33 c0</td>
<td>c6 9d b8 15</td>
</tr>
<tr>
<td></td>
<td>e8 c0 50 01</td>
<td>9b ba 53 7c</td>
<td>7c 9b ba 53</td>
<td>ad 68 8e b0</td>
<td>f8 87 bc bc</td>
</tr>
</tbody>
</table>
Appendix C – Example Vectors

This appendix contains example vectors, including intermediate values – for all three AES key lengths (\(N_k = 4, 6, \text{ and } 8\)), for the Cipher, Inverse Cipher, and Equivalent Inverse Cipher that are described in Sec. 5.1, 5.3, and 5.3.5, respectively. Additional examples may be found at [1] and [5].

All vectors are in hexadecimal notation, with each pair of characters giving a byte value in which the left character of each pair provides the bit pattern for the 4 bit group containing the higher numbered bits using the notation explained in Sec. 3.2, while the right character provides the bit pattern for the lower-numbered bits. The array index for all bytes (groups of two hexadecimal digits) within these test vectors starts at zero and increases from left to right.

Legend for CIPHER (ENCRYPT) (round number \(r = 0 \text{ to } 10, 12 \text{ or } 14\)):

- **input**: cipher input
- **start**: state at start of round\([r]\)
- **s_box**: state after SubBytes()
- **s_row**: state after ShiftRows()
- **m_col**: state after MixColumns()
- **k_sch**: key schedule value for round\([r]\)
- **output**: cipher output

Legend for INVERSE CIPHER (DECRYPT) (round number \(r = 0 \text{ to } 10, 12 \text{ or } 14\)):

- **iinput**: inverse cipher input
- **istart**: state at start of round\([r]\)
- **is_box**: state after InvSubBytes()
- **is_row**: state after InvShiftRows()
- **ik_sch**: key schedule value for round\([r]\)
- **ik_add**: state after AddRoundKey()
- **ioutput**: inverse cipher output

Legend for EQUIVALENT INVERSE CIPHER (DECRYPT) (round number \(r = 0 \text{ to } 10, 12 \text{ or } 14\)):

- **iinput**: inverse cipher input
- **istart**: state at start of round\([r]\)
- **is_box**: state after InvSubBytes()
- **is_row**: state after InvShiftRows()
- **im_col**: state after InvMixColumns()
- **ik_sch**: key schedule value for round\([r]\)
- **ioutput**: inverse cipher output

C.1 AES-128 (\(N_k=4, N_r=10\))

**PLAINTEXT**: 00112233445566778899aabbccddeeff
**KEY**: 000102030405060708090a0b0c0d0e0f

**CIPHER (ENCRYPT)**:
round[0].input 0011233445566778899aabbccddeeff
round[0].k_sch 00102030405060708090a0b0c0d0e0f
round[1].start 00102030405060708090a0b0c0d0e0f
round[1].s_box 63cab7040953d051cd60e7ba70e18c
round[1].s_row 6353e0c80960e04c70b751bacade07
round[1].m_col 5f7264155f5bc92f7be3b291db9f91a
round[1].k_sch d6aa74fdd2af72fadaa678f1d6ab76fe
round[2].start 89d810e885ace682d1843d8cb1286f4
round[2].s_box a761ca9b97be8b45d8ad1a611fc97369
round[2].s_row a7be1a6997ad739bd8c9a451f6186b1
round[2].m_col ff87968431d6a51645151fa773ad009
round[2].k_sch b692cf0b643dbdf1be9bc506830b3fe
round[3].start 491559f55e5d7a0daca94fa1f0a63f7
round[3].s_box 6359cb73fcd90ee05774222dc067fb68
round[3].s_row 639d2268fc74fbd7537577cbe0c0590e2d
round[3].m_col 4c91e66f77f0762c3f86e534df256
round[3].k_sch b6ff744ed2c29bf6c90cb40469bf41
round[4].start fa63a2825b339c940668a3157244d17
round[4].s_box 6385b797f5e3d99be4787547d691
round[4].s_row 63d7f7fbc95353e30f96e32bcfd058dfe
dround[4].m_col 2d6d7ef03f3e334039602dd5bf6b12c7
round[5].start 491559f55e5d7a0daca94fa1f0a63f7
round[5].s_box 6359cb73fcd90ee05774222dc067fb68
round[5].s_row 639d2268fc74fbd7537577cbe0c0590e2d
round[5].m_col 2d6d7ef03f3e334039602dd5bf6b12c7
round[6].start c62fe109f75eedc3cc7935d849c9f5d
round[6].s_box b415f8016858552e2bb6124c5f998a4c
round[6].s_row b458124c68b68a014b99f8e2e5f15554c
round[6].m_col c57e1c159a9bd286f05f4be098c63439
round[7].start c62fe109f75eedc3cc7935d849c9f5d
round[7].s_box b415f8016858552e2bb6124c5f998a4c
round[7].s_row b458124c68b68a014b99f8e2e5f15554c
round[8].start 4f9701ae35fe28c4404df44e9c026
round[8].s_box d8176c0f79c4300ab45594add66ff41f
round[8].s_row 4f9701ae35fe28c4404df44e9c026
round[9].start 7ad5fda789ef4e272bca100b3d9ff59f
round[9].s_box 13111d7fe3944a17f307a78b4d2b3b0c5
round[10].output 69c4e0d86a7b0430d8c17b8070b4c55a

INVERSE CIPHER (DECRYPT):
round[0].iinput 69c4e0d86a7b0430d8c17b8070b4c55a
round[0].ik_sch a9f7e4e023020f61bf2ccf2353c21c7
round[0].istart 7ad5fda789ef4e272bca100b3d9ff59f
round[1].iistart 13111d7fe3944a17f307a78b4d2b3b0c5

36
EQUIVALENT INVERSE CIPHER (DECRYPT):

round[ 0].iinput 69c4e0d86a7b0430d8cdb78070b4c55a
round[ 0].ik_sch 13111d7fe3944a17f307a78b4d2b30c5
round[ 1].istart 7ad5fda789ef4e272bca100b3d9ff59f
round[ 1].is_row 6353e08c0960e104cd7b751bacd0e7
round[ 1].is_box 63cab7040953d051cd60e0e7a70e18c
round[ 1].ik_sch 00102030405060708090a0b0c0d0e0f
round[ 1].ioutput 00112233445566778899aabccddeeff

round[ 1].is_row 7a9f102789d5f05b2beffdf93dca4ea7
round[ 1].is_box bd6e7c3d2f2b5779e0b61216e8b10b689
round[ 1].ik_sch 549932d1f08557681093ed9cbe2c974e
round[ 1].ik_add e9f74ec023020f61bf2ccf2353c21c7
round[ 2].istart 54d990a16ba9ab596bbf4e0a111702f
round[ 2].is_row 5411f4b56bd9700e96a0902f1ab9b9a1
round[ 2].is_box fde3bad205e5d073457946ef1fe37f1
round[ 2].ik_sch 47438735a41c65b9e016bfa4aebf7ad2
round[ 2].ik_add baa03de7a1f9b56ed5512cba5f414d23
round[ 3].istart 3e1c222c06fcfb7f68da85067f6170495
round[ 3].is_row 3e175076b61c04678dfc2295f6a8bfec0
round[ 3].is_box d1876c0f79c4300ab45594add66ff41f
round[ 3].ik_sch 14f9701ae35fe28c440af4d4ea9c026
round[ 3].ik_add c57e1c159a9bd86f05ff4e098e63439
round[ 4].istart b458124c68b68a014b9f982e5f15554c
round[ 4].is_row b415f8016858552e4b6b1245c5998a4c
round[ 4].is_box c62fe109f75e6edc3c79395d84f9cf5d
round[ 4].ik_sch 5e390f7df7a69296a7553dc1a31f6b
round[ 4].ik_add 9816ee7400f8755b6b049c8e5a0d36
round[ 5].istart e8db6901477d4653ff7fe2e747dd4f
round[ 5].is_row e847f56514dadde23f77b64fe7f74d90
round[ 5].is_box c81677bc97abc93b2507992b0261996
round[ 5].ik_sch 3caaa3e8a999fdeb50f3af5ad6f22aa
round[ 5].ik_add f4b4cd5432e5540d75f1d6c51dd03b3c
round[ 6].istart 3633950f9b539269f2c092d44060d23
round[ 6].is_row 36400926f9336d2d9fb59d23c42c3950
round[ 6].is_box 2472402369b63fa6ed2753288425b6c
round[ 6].ik_sch 47f7f7b9c5353e03f96c32bce0d585fd
round[ 6].ik_add 6385b79f5c538df997be478e7547d691
round[ 7].istart 2d6d7ef03f3e334039602dd5bfb12c7
round[ 7].is_row 2dfb02343f6612d9337ec75b36e3f0
round[ 7].is_box fa636a2825b339c940668a3157244d17
round[ 7].ik_sch b6ff744ed2c2cb9f6c950cf0469bf41
round[ 7].ik_add 4cc9e166f771f0762c3f86e534df256
round[ 8].istart 3bd92268fc74fbb735767cbe0c0590e2d
round[ 8].is_row 359bc7973fcd90ee05774222d6c076f68
round[ 8].is_box 4915598f55e5d7a0daca94fa1f0a63f7
round[ 8].ik_sch b692cf0b6344d0f1be9bc5006830b3fe
round[ 8].ik_add ff8796843186a51645151a773aad009
round[ 9].istart a7be1a6997ad739bd8c9ca451f618b61
round[ 9].is_row a761ca9b7be845d8ad1a611fc97369
round[ 9].is_box 89d810e8855ace682d1843d8c1b28e4
round[ 9].ik_sch d6aa74fdd2af72fadaa678f1d6ab76fe
round[ 9].ik_add 5f72641557f5bc92f7be3b92d99f91a
round[ 10].istart 3635e08c0960e104cd7b751bacd0e7
round[ 10].is_row 63cab7040953d051cd60e0e7a70e18c
round[ 10].is_box 00102030405060708090a0b0c0d0e0f
round[ 10].ik_sch 00102030405060708090a0b0c0d0e0f
round[ 10].ioutput 00112233445566778899aabccddeeff

37
C.2 AES-192 (Nk=6, Nr=12)

PLAINTEXT: 00112233445566778899aabbccddeeff
KEY: 000102030405060708090a0b0c0d0e0f011121314151617

CIPHER (ENCRYPT):

round[ 0].input 00112233445566778899aabbccddeeff
round[ 0].k_sch 000102030405060708090a0b0c0d0e0f011121314151617
round[ 1].start 000102030405060708090a0b0c0d0e0f0
INVERSE CIPHER (DECRYPT):

round[0].iinput dda97ca4864cdfe06eaf70a0ec0d7191
round[0].ik_sch a4970a331a78dc09c418c271e3a41d5d
round[1].istart 793e76979c3403e9aab72d10fa96ccc
round[1].is_row 79a9b2e99c36c1aa3476cc0fb70397
round[1].is_box afb73ebe1cd185162280f27fb20d585
round[1].ik_sch de601e7827bdcf2ca223800fd8a6ed3a
round[1].ik_add 71d72093356e777dc00b8f28238e0fb7
round[2].irow c494bfff6e26232ab4bb5dc4e6f6e69dd
round[2].is_box c4cedcabe694694eb4b23f4bb6bb522fa
round[2].iinput 88ec930ef5e4b66cc32f4c906d29414
round[2].ik_sch 859f5f237a8d5a30e02952b6e6d53a
round[2].ik_add 0d73cc2d86aabe8b0c2fd2db9b83d422e
round[3].istart d3e3705907a1a208d1c371e8c6f5fb5
round[3].is_row d36f3720970ef1e8d7a37b58c1c1a05
round[3].is_box a906b254968af49b4b2db2d0c44336
round[3].ik_sch df7e0e876a2ff66808fc842f9ddcc154
round[3].ik_add 7478bcdce8a50b81d43279a90918262
round[4].istart 406c501076d70066e17057ca09fc77b7f
round[4].is_row 40fc5766667bcaed17507f9700010
round[4].is_box 7255dad30f8b030200e66cb4d0527c
round[4].ik_sch 1ea30729a95309167c4397e77f12051e
round[4].ik_add 6cf5edf996eb0a069c4ef21bdfbfc25762
round[5].irow fe7c7e71fe7f807047b95193f67b8e4eb
round[5].is_row fe7b5170fe7c8e934777fe4bf6b98071
round[5].is_box 0c0370d00c01e2622166b8acc6db3a2c
round[5].ik_sch e510976183519b6934157c9ea351f1e0
round[5].ik_add e913e7b1f85f07d42b27ef652758acbbcc
round[6].istart 85e5c8042ff8614549abca17b277272df
round[6].is_row 8572a1542fe5727b9e86c8d2f27bc1404
round[6].is_box 671e1fd4e2a1e03dfcb1e3fd789b30
round[6].ik_sch f5018572794487e76b1c6a87f33e3c
round[6].ik_add 921f748fd96e937d6227725baba50c
round[7].istart cd54c728386d0c554d4727e90c9a465
round[7].is_row cdc972c53854a47e5d4c7659004c028
round[7].is_box 80121e0776fd1d8a8d8c31bc965d1fee
round[7].ik_sch 2ab54bb4a028f662e3a95d66410c08
round[7].ik_add aaa755b34cfe57cef6f98ef01c13e6
round[8].irow 93faa123c2903f4743e4dd83431692de
round[8].is_row 9316dd47c2fa92834390a1de43e43f23
round[8].is_box 22ffcc916a81474416496f19c64ae2535
round[8].ik_sch 58e151ab04a2a5557effb5416255080c
round[8].ik_add 7a1e98bdac6b6114a1e944dd06eb2d3e
round[9].istart 68c808ed0abbdd2bc642ef55244ae878
round[9].is_row 684af5bc0acce85564bb0878242ed2ed
round[9].is_box f75c7778a327c8ed8cfebcfca16c37f53
round[9].ik_sch 40f949b31cababd4d48f0438b107b342
round[9].ik_add b7a53ecbbf97d75a0f40efc79b674cc11
round[10].istart 1fb5430ef0accf66a4370cde377792c
round[10].is_row 1f770c64f0b579deaac432c37d7cf0e
round[10].is_box cb02818c17d2af9c62a64428bb25fd7
round[10].ik_sch 544afe55847f0f4a856e2e95c43f4fe
round[10].ik_add 9f487f794f955f662af86ad7f1ab29
round[11].istart 84e1dd691a41d76f792d389783fbac70
EQUIVALENT INVERSE CIPHER (DECRYPT):

round[0].iinput  dda97ca4864cdfe06eaf70a0ec0d7191
round[0].ik_sch  a4970a331a78dc09c418c271e3a41d5d
round[1].istart  793e76979c3403e9aab7b2d10f9a96ccc
round[1].is_box  afd10f851c28d5eb2203e51fbb7b827
round[1].is_row  afb73ee1cd1b85162280f27fbb20d585
round[1].im_col  122a02f7242ac8e20605afce51c7264
round[1].ik_sch  d6beb0d0c209e494db073803e021bb9
round[2].ioutput  c494bffa6e2322ab4bb5dc4e6fcee69dd
round[2].is_row  88e7f414f532940eccdc293b606ece4c9
round[2].istart  88ec930ef5e4b6ccc32f4c906d29414
round[2].im_col  5cc7aecece3c872194ae5ef8309a933c7
round[2].is_box  8fb9999c973b26839c7f9d8985c68c72
round[2].ik_sch  d37e3705907a1a2081c371e8c6fbbf5
round[3].istart  a98ab23696bd4354bc4e2af006f4d2
round[3].im_col  a906b25a968af4e94bbdb2dfc44336
round[3].is_row  b7113ed134e85489b208665b1d4b2c3b
round[3].is_box  f776d6c1423f54af5378317f14b75744
round[3].ik_sch  406c501076d70066e17057ca09fc7bf7
round[4].istart  72b86c70c0f0d52d3e0d0da104055036b
round[4].is_box  7255dad30f8031000e66c6b40d0527c
round[4].im_col  ef3b1be1b90e64dcb79f1e0a707fbb
round[4].is_row  d1647659047cf663b90e8e8d0f0bf1f0
round[4].ik_sch  fe7c7e7fe7f807047b95193f67b8e4b
round[5].is_row  0c018a20c06b3ad016d7022603e6cc
round[5].iinput  0c0370d00c01e622166b8accd6d3a2ac
round[5].im_col  592460b248832b2952e0831923048f1
round[5].is_box  ddc1a8b667053f7dccc5194ab5423a2e
round[6].istart  85ec50844f6814549e6ca177b277272df
round[6].im_col  672a01b30c9bafdf78f1033d1e1eef
round[6].is_box  671ef1fd4e2a1e03d4d0f0c3f768b30
round[6].is_row  0b8a7783417ae3a1f942dc0c641a7ce
round[6].ik_sch  c6deb0ab791e2364a4055f5e68303ab
round[7].istart  cd54c7283864c0c554d4727e90c9a465
round[7].im_col  80fd31ee768c1f0785d1e8a96121d7bc
round[7].is_row  80121e0776f71d8a8d8c31b965d1fee
round[8].iinput  4e1eddf9301d6352c9ad769ef8d20515
round[8].im_col  dd1b7caedf2825c518a49ab1d6bc497cb
round[8].is_row  93faa123c2903f4743e4dd83431692de
round[8].is_box  2214f132a896251664a39c9414ff3749c
round[8].ik_sch  22ff916a8147441649f19c64ae2532
round[9].istart  1008ffe53b36ee6af27b42549b8a7bb7
round[9].im_col  78c4f7083818d3cd69565b701bc093cf
round[9].is_row  68cco8ed0abb2bc642ef555244ae878
round[9].is_box  f727bf53a3fe7f788cc377eda65cc8c1
round[9].ioutput  f75c7778a327c8ed8cfebf1a6c37f53
round[9].ik_sch  7f69ac1ed939ebacc8e3c3eb12e159e3

41
C.3 AES-256 ($N_k=8$, $N_r=14$)

PLAINTEXT: 00112233445566778899aabbccddeeff

KEY: 000102030405060708090a0b0c0d0e0f10111213141516171819a1b1c1d1e1f

CIPHER (ENCRYPT):

```
round[ 0].input 00112233445566778899aabbccddeeff
round[ 0].k_sch 00102030405060708090a0b0c0d0e0f
round[ 1].start 00102030405060708090a0b0c0d0e0f
round[ 1].s_box 63cab7040953d051cd60e0e7ba70e18c
round[ 1].s_row 6353e08c0960e104cd70b751bacad0e7
round[ 1].m_col 5f72641557f5bc92f7be3b291dbad91a
round[ 1].k_sch 000102030405060708090a0b0c0d0e0f
round[ 2].start 4f63760643e08a85fa7a213201a4e705
round[ 2].s_box 84fb386f1ae19cd7f5cfd37c49946b
round[ 2].s_row 84efdb615ac946df4938977cbac23
round[ 2].m_col bd2a395d2b6ac438d192443e615da195
round[ 2].k_sch a573c29fa176c49897fe393a572c09c
round[ 3].start 1859fbc28a1c0078ed8aad42f6109
round[ 3].s_box adbcb0f257963e0557e951c15ef01
round[ 3].s_row ad9c7e017e55f5f25bc150fe01ccb6395
round[ 3].m_col 810dce0cc9db8172b3678c88a1b5bd
round[ 3].k_sch 1651a8cd0244bed1a5da4c10640ade
round[ 4].start 975c66c1cb9c05a93a8dfeb8e10f63
round[ 4].s_box 884a33781f752c3d80349e19f876fb
round[ 4].s_row 88db34f1807678d3f833c2194a759e
round[ 4].m_col b2822d81aebb6fb275fafa103a078c0033
round[ 5].start a87d4f0f11b68a68ed5f03c1567
round[ 5].s_box 1c05f271a417e04ff921c5c104701554
round[ 5].s_row 9c6b89a349f0e18499fda678f2515920
round[ 6].start aeb65ba974e0f822d73f57d7b4d64c877
round[ 6].s_box 6def1f486fa54f9275f8e53738b518d
round[ 6].s_row c3577ae11b457b0a2c7b2d8a8dc9fa
round[ 7].start 2e5baccf8af6ea973ac67a34c2868e2d
round[ 7].s_box 2e6ea2daef6e83a86ace7c25ba934
round[ 7].s_row b951c33c02e9bd29ae25c01f0a8cc7
round[ 8].start c656827fc9a79917f294ce6c6d5598b
round[ 8].s_box 7f074143cb4e243ec10c815d8375d54c
round[ 8].s_row d2c5831a1f2f36b278fe0c4ce9d0329
```
round[7].s_row d22f0c291ffe031a789d83b2ecc5364c
round[7].m_col ebb19e1c3ee7c9e87d7535e9ed6b9144
round[7].k_sch 3de23a755247757e727bf9eb45407c3f9
round[8].start d653a4696ca0bc0f5acaab5db96c5e7d
round[8].s_box f6ed49f950e06576be7462c565058ff
round[8].s_row f6e062ff507458f9be50497656ed654c
round[8].m_col 5174c8669da98435a8b3e62ca974a5ea
round[8].k_sch 0bdc905fc27b0948ad5245a41871c2f
round[9].start 5aa858395fd28d70d05e1a38868f3bc9c
round[9].s_box bec26a12c5bf55df6f680ac4450d56a6
round[9].s_row beb50a6cff856126bd6aff45c25dc4
round[9].m_col 0f77ee31d2ccadcc05430a83f4ef9ac3
round[9].k_sch 45fa5e6017bd2387300d4d33640a820a
round[10].start 4a824851c57e7e47643de50c2aef8e0c9
round[10].s_box d61352d1af3f304327d9f6e509b7dd
round[10].s_row d6f39d9da279bd1430522ae5f13f3e
round[10].m_col bd8e0a7480c4e4637f11ce9331233
round[10].k_sch 7ccff71cbe4fe5413e6bfbfd261a7df
round[11].start c14907f6ca38a0709ea3135b25e5ec
round[11].s_box 783bc54274ae280e0511eacc7e200d5ce
round[11].s_row 78e2acce741ed54251005e0e23b80c7
round[11].m_col af869041d6e1d3d87e5fbddd5c89013
round[11].k_sch f01afafeeaa82979d7a5644ab3af6e40
round[12].start 5f9c6abfba634aa5049fa766677653
round[12].s_box cfde02084b418ac5309db5c33838ed
round[12].s_row cfbd4bedf4093808538502ac33de185c
round[12].m_col 7427fae4d8a695269ce83d315b0e0392b
round[12].k_sch 2541fe719bf500258813b955a721c0a
round[13].start 51660495435935014fb86e401922521
round[13].s_box d133f2a1a2ed2a7ba0f4f4697c4f3f4fd
round[13].s_row d1ed44fdaa0f3f2a4f4ff27b7c332a69
round[13].m_col 2c2b8a120306fb154ab712c75ee0da40f
round[13].k_sch 4e5a6069a9f24fe07e572baacdf8cdea
round[14].start 627bce9999d5aaac945ecf423f56da5
round[14].s_box aa218b56ee5ebeacdd6cebf26e63c06
round[14].s_row aa5e0e06e6e3c56dde68bac2621bebf
round[14].k_sch 24fc79ccbf0979e9371ac23c668de36
round[14].output 8ea2b7ca56745bfeafcc49004b496089

INVERSE CIPHER (DECRYPT):

round[0].iinput 8ea2b7ca56745bfeafcc49004b496089
round[0].ik_sch 24fc79ccbf0979e9371ac23c668de36
round[1].istart aa5e06e6e3c56dde68bac2621bebf
round[1].is_row aa218b56ee5ebeacdd6cebf26e63c06
round[1].is_box 627bce9999d5aaac945ecf423f56da5
round[1].ik_sch 4e5a6069a9f24fe07e572baacdf8cdea
round[1].ik_add 2c21a820306f154ab712c75ee0da40f
round[2].istart d1ed44fdaa0f3f2a4f4ff27b7c332a69
round[2].is_box d133f2a1a2ed2a7ba0f4f4697c4f3f4fd
round[2].is_row 51660495435935014fb86e401922521
round[2].ik_sch 2541fe719bf500258813b955a721c0a
round[2].ik_add 7427fae4d8a695269ce83d315b0e0392b
round[3].istart cfbd4bedf4093808538502ac33de185c
round[3].is_row cfde02084b418ac5309db5c33838ed
round[3].is_box 5f9c6abfba634aa5049fa766677653
round[3].ik_sch f01afafeeaa82979d7a5644ab3af6e40
round[3].ik_add af869041d6e1d3d87e5fbddd5c89013
EQUIVALENT INVERSE CIPHER (DECRYPT):
44
round[12].istart  ad9c7e017e55ef25bc150fe01ccb6395
round[12].is_box   181c8a098aed61c2782ffba0c45900ad
round[12].is_row   1859fbc28a1c00a078ed8aad42f6109
round[12].im_col   aec9bda23e7fd8aff96d74525cde4e7
round[12].ik_sch   2a2840c924234cc026244cc5202748c4
round[13].istart   84e1fd6b1a5c946fdef4938977cfbac23
round[13].is_box   4fe021054a7e706efa476850163aa32
round[13].is_row   4f63760643e0aa85efa7213201a4e705
round[13].im_col   794cf891177bfdd1df67a744ac9c4f6
round[13].ik_sch   1a1f181d1e1b1c191217101516131411
round[14].istart   6353e08c0960e104cd70b751bacad0e7
round[14].is_box   0050a0f04090e03080d02070c01060b0
round[14].is_row   00102030405060708090a0b0c0d0e0f
round[14].ik_sch   00012030405060708090a0b0c0d0e0f
round[14].ioutput  00112233445566778899aabbccddeeff
Appendix D - References


---

4 A complete set of documentation from the AES development effort – including announcements, public comments, analysis papers, conference proceedings, etc. – is available from this site.