1. Introduction

This standard specifies the Rijndael algorithm, 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.

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 *Nb* = 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.

Nb Number of columns (32-bit words) comprising the State. For this standard, ***Nb***= 4.

Nk Number of 32-bit words comprising the Cipher Key. For this standard, ***Nk*** = 4

Nr Number of rounds, which is a function of ***Nk*** and ***Nb*** (which is fixed). For this standard, ***Nr*** = 10.

Rcon[] The round constant word array.

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

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

SubBytes() Transformation in the Cipher that processes the State using a nonlinear byte substitution table (S-box) that operates on each of the State bytes independently.

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.

3. Cipher

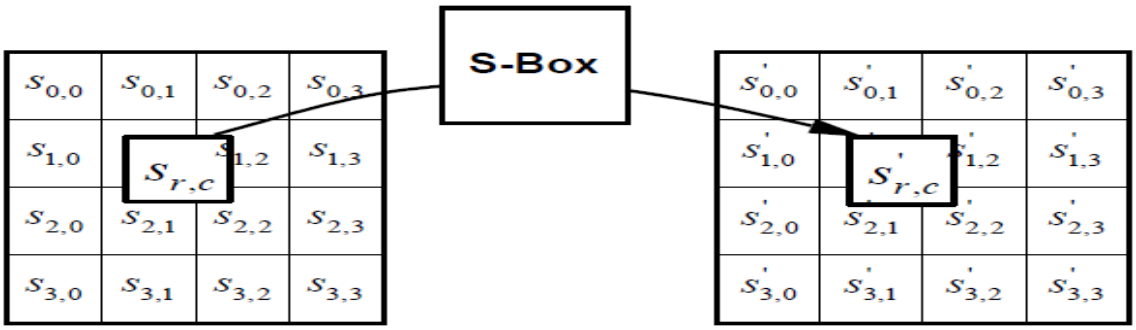
At the start of the Cipher, the input is copied to the State array using the conventions described in previous section. After an initial Round Key addition, the State array is transformed by implementing a round function 10, 12, or 14 times (depending on the key length), with the final round differing slightly from the first *Nr* -1 rounds. 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.The Cipher is described in the pseudo code. The individual transformations - SubBytes(), ShiftRows(), MixColumns(), and AddRoundKey() – process the State and are described in the following subsections. All *Nr* rounds are identical with the exception of the final round, which does not include the MixColumns() transformation.

```
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])
     for round = 1 step 1 to Nr-1 SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        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
```

Pseudo Code for the Cipher.

3.1 SubBytes()Transformation:

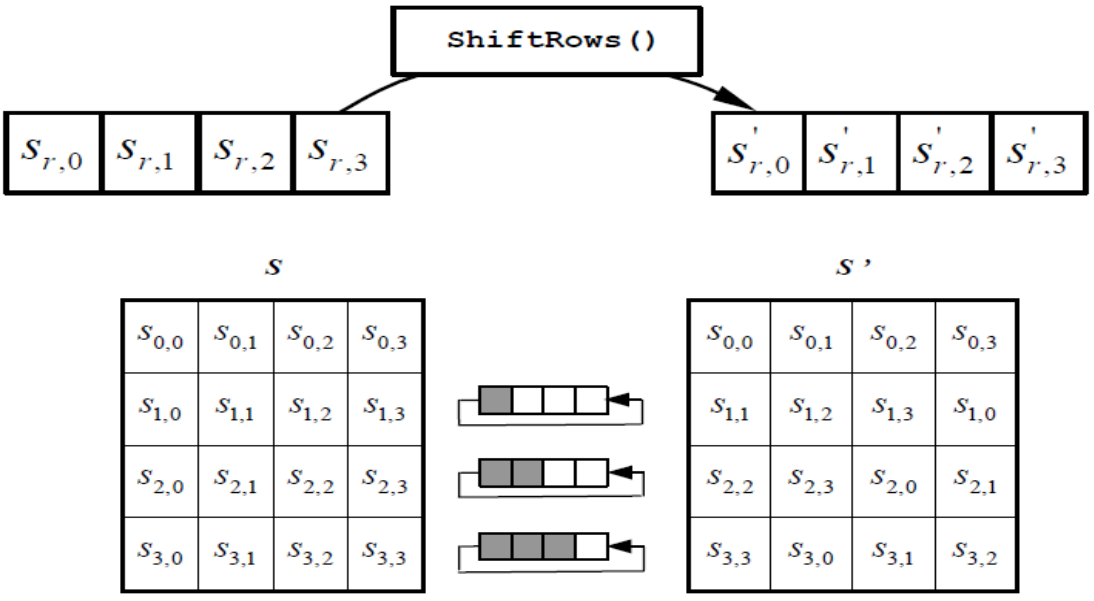
The SubBytes() transformation is a non-linear byte substitution that operates independently on each byte of the State using a substitution table (S-box).



SubBytes() applies the S-box to each byte of the State.

3.2 ShiftRows() Transformation:

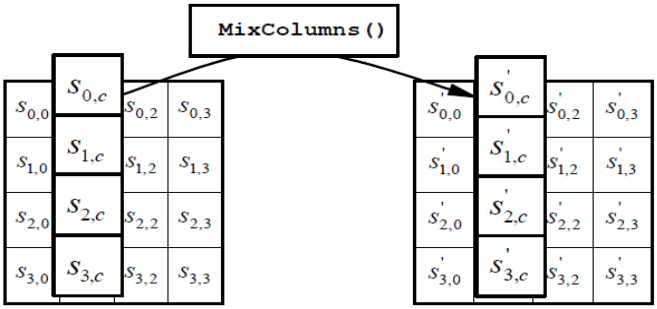
In the ShiftRows() transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, $r = 0$, is not shifted.



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

3.3 MixColumns() Transformation:

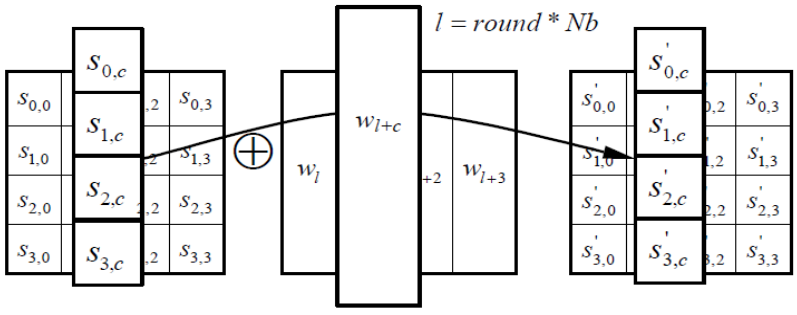
The MixColumns() transformation operates on the State column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x)$, given by $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.



MixColumns() operates on the State column-by-column.

3.4 AddRoundKey() Transformation:

In the 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. Those *Nb* words are each added into the columns of the State.



AddRoundKey() XORs each column of the State with a word from the key schedule.

4. 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* (*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 [*wi*], with *i* in the range 0 < *i* < *Nb*(*Nr* + 1). SubWord() is a function that takes a four-byte input word and applies the S-box to each of the four bytes to produce an output word. The function 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, 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), (note that *i* starts at 1, not 0).

```
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
```

Pseudo Code for Key Expansion.

5. Inverse Cipher

The Cipher transformations in Sec. 4 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.

```
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])

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

    InvShiftRows(state)

    InvSubBytes(state)

    AddRoundKey(state, w[0, Nb-1])

out = state

end
```

Pseudo Code for the Inverse Cipher.

5.1 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 $Nb\text{-}shift(r, Nb)$ bytes.

5.2 InvSubBytes() Transformation

InvSubBytes() is the inverse of the byte substitution transformation, in which the inverse S-box is applied to each byte of the State.

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

5.4 Inverse of the AddRoundKey() Transformation

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

6. References:

1) www.csrc.nist.gov/publications/fips/fips197.