THE ADVANCED ENCRYPTION STANDARD (AES)

1. Preliminaries

1.1. **Operations on Bytes.** Consider a byte $b = (b_7, b_6, \ldots, b_1, b_0)$ (an 8-bit vector) as a polynomial with coefficients in $\{0, 1\}$:

$$b \mapsto b(x) = b_7x^7 + b_6x^6 + \cdots + b_1x + b_0 \ .$$

RIJNDAEL makes use of the following operations on bytes, interpreting them as polynomials:

(1) Addition: polynomial addition by taking XOR of coefficients.

$$\begin{array}{cccccccccc} & b_7x^7 & + & b_6x^6 & +\cdots+ & b_1x & + & b_0 \\ + & c_7x^7 & + & c_6x^6 & +\cdots+ & c_1x & + & c_0 \\ \hline & (b_7 \oplus c_7)x^7 & + & (b_6 \oplus c_6)x^6 & +\cdots+ & (b_1 \oplus c_1)x & + & (b_0 \oplus c_0) \end{array}$$

The sum of two polynomials taken in this manner yields another polynomial of degree 7. In other words, component-wise XOR of bytes is identified with this addition operation on polynomials.

(2) Multiplication: polynomial multiplication (coefficients are in $\{0, 1\}$) modulo $m(x) = x^8 + x^4 + x^3 + x + 1$ (remainder when dividing by $m(x)$ — analogous to modulo arithmetic with integers). The remainder when dividing by a degree 8 polynomial will have degree ≤ 7. Thus, the "product" of two bytes is associated with the product of their polynomial equivalents modulo $m(x)$.

(3) Inverse: $b(x)^{-1}$, the inverse of $b(x) = b_7x^7 + b_6x^6 + \cdots + b_1x + b_0$, is the degree 7 polynomial with coefficients in $\{0, 1\}$ such that

$$b(x)b(x)^{-1} \equiv 1 \pmod{m(x)} \ .$$

Note that this is completely analogous to the case of integer arithmetic modulo n. In this case the "inverse" of the byte $b = (b_7, b_6, \ldots, b_1, b_0)$ is the byte associated with the inverse of $b(x) = b_7x^7 + b_6x^6 + \cdots + b_1x + b_0$.

By associating bytes with polynomials, we obtain the above three operations on bytes. RIJNDAEL uses inverse as above in the ByteSub operation.

$\mathbb{F}_{2^8}$ is the set of 256 bytes viewed as polynomials, together with the operations described above.

1.2. **4-byte Vectors.** In the MixColumn operation of RIJNDAEL, 4-byte vectors are considered as degree 3 polynomials with coefficients in $\mathbb{F}_{2^8}$. That is, the 4-byte vector (a_3, a_2, a_1, a_0) is associated with the polynomial

$$a_3x^3 + a_2x^2 + a_1x + a_0,$$

where each coefficient is a byte viewed as an element of $\mathbb{F}_{2^8}$ (addition, multiplication, and inversion of the coefficients is performed as described above). We have the following operations on these polynomials:

(1) addition: component-wise "addition" of coefficients (as described above)
(2) multiplication: polynomial multiplication (addition and multiplication of coefficients as described above) modulo $M(x) = x^4 + 1$. Result is a degree 3 polynomial with coefficients in $\mathbb{F}_{2^8}$.

In MixColumn, the 4-byte vector (a_3, a_2, a_1, a_0) is replaced by the result of multiplying $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ by the fixed polynomial

$$c(x) = 03x^3 + 01x^2 + 01x + 02$$

and reducing modulo $x^4 + 1$. The coefficients of $c(x)$ are given as bytes in hex notation.

2. The Rijndael Algorithm

Rijndael (developed by Daemen and Rijmen):

- designed for block sizes and key lengths to be any multiple of 32, including those specified in the AES ($n = 128$, $m = 128, 192, 256$)
- iterated cipher: number of rounds, N_r depends on the key length. $N_r = 10$ for $m = 128$, $N_r = 12$ for $m = 192$, and $N_r = 14$ for $m = 256$ (see p. 14 of NIST document).
- $\mathbb{F}_{2^8} = \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$ used for non-linear byte operations.
- the algorithm operates on a 4×4 array of bytes called the *state*:

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

  The dimensions of the state depend on the block size.
- the key is expanded into $N_r + 1$ *round keys*, where each round key consists of the same number of bytes as the state.

The Rijndael algorithm (given plaintext M) proceeds as follows (p. 9):

(1) Initialize `State` with M :

$$\begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \leftarrow \begin{array}{|c|c|c|c|} \hline m_0 & m_4 & m_8 & m_{12} \\ \hline m_1 & m_5 & m_9 & m_{13} \\ \hline m_2 & m_6 & m_{10} & m_{14} \\ \hline m_3 & m_7 & m_{11} & m_{15} \\ \hline \end{array}$$

where M consists of the 16 bytes $m_0, m_1, \ldots, m_{15}$.

(2) Perform AddRoundKey, which XOR's the first `RoundKey` with `State`.

(3) For each of the first $N_r - 1$ rounds:
   - Perform SubBytes on `State` (using a substitution, or S-box, on each byte of `State`),
   - Perform ShiftRows (a permutation) on `State`,
   - Perform MixColumns (a linear transformation) on `State`,
   - Perform AddRoundKey.

(4) For the last round:
   - Perform SubBytes,
   - Perform ShiftRows,
   - Perform AddRoundKey.

(5) Define the ciphertext C to be `State` (using the same byte ordering).

Note: Rijndael is a product cipher: each round contains subkey mixing (AddRoundKey), substitution (SubBytes), and a permutation (ShiftRows and MixColumns).

2.1. **The SubBytes Operation.** (p.15) Each byte of `State` is substituted (independently). Can be implemented via table lookup (memory permitting), but is described algebraically. Let ϕ be the function mapping bytes to elements of $\mathbb{F}_{2^8}$ defined by

$$\phi : (a_7 a_6 \dots a_0) \mapsto \sum_{i=0}^{7} a_i x^i, a_i \in \mathbb{F}_2 = \{0,1\} \ .$$

Then:

$$\text{SubBytes}(a) = \phi^{-1}\left[(x^4+x^3+x^2+x+1)\phi(a)^{-1} + (x^6+x^5+x+1) \bmod (x^8+1)\right] \ .$$

This operation can be performed using the following steps:

(1) $z = \phi(a)$ (field representation of the byte a)
(2) $z = z^{-1}$ (take the inverse in $\mathbb{F}_{2^8}$)
(3) $b = \phi^{-1}(z)$ (map the field element z to the byte b)
(4) Output the byte b' using the following affine transformation:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Note that $b' = (b'_7 b'_6 \dots b'_0)$ where

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

and $c = (11000110)$.

The inverse of SubBytes (called InvSubBytes, p. 22) is defined by

$$\text{InvSubBytes}(a) = \phi^{-1}\left[((x^6+x^3+x)\phi(a) + (x^2+1) \bmod (x^8+1))^{-1}\right] \ .$$

2.2. **The ShiftRows Operation.** (p. 17) Shifts the rows of `State` by 0, 1, 2, or 3 cells to the left:

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$		$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$\to$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$		$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$		$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

The inverse operation InvShiftRows (p. 21) applies right shifts instead of left shifts.

2.3. **The MixColumns Operation.** (p. 17) Consider each column of `State` as a four-term polynomial with coefficients in $\mathbb{F}_{2^8}$. For example:

$$(s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}) \mapsto s_{3,0}y^3 + s_{2,0}y^2 + s_{1,0}y + s_{0,0} = col_0(y) \ .$$

Let $a(y) = (x+1)y^3 + y^2 + y + (x)$ be fixed. Then the MixColumns operation replaces each column of `State` via

$$col_i(y) \leftarrow a(y)col_i(y) \pmod{y^4+1}, \quad i = 0,1,2,3 \ .$$

Note: MixColumns can also be described as a linear transformation applied to each column of `State`, i.e., multiplying each 4-element column vector by a 4×4 matrix with coefficients in $\mathbb{F}_{2^8}$.

The inverse (called InvMixColumns, p. 23) is given by

$$col_i(y) \leftarrow a(y)^{-1}col_i(y) \pmod{y^4+1}, \quad i = 0,1,2,3$$

and can also be described as a linear transformation.

2.4. ADDROUNDKEY **and the Key Schedule.** In ADDROUNDKEY (p. 23), each column of `State` is XORed with one word of the round key:

$$\begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \leftarrow \begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|}\hline w_{0,i+0} & w_{0,i+1} & w_{0,i+2} & w_{0,i+3} \\ w_{1,i+0} & w_{1,i+1} & w_{1,i+2} & w_{1,i+3} \\ w_{2,i+0} & w_{2,i+1} & w_{2,i+2} & w_{2,i+3} \\ w_{3,i+0} & w_{3,i+1} & w_{3,i+2} & w_{3,i+3} \\ \hline \end{array}$$

Here $w_{i+0} = (w_{0,i+0}, w_{1,i+0}, w_{2,i+0}, w_{3,i+0})$ is the first round key for round i, made up of four bytes.

ADDROUNDKEY is clearly it's own inverse.

Consider 128-bit Rijndael. There are 10 rounds plus one preliminary application of ADDROUNDKEY, so the key schedule must produce 11 round keys, each consisting of four 4-byte words, from the 128-bit key (16 bytes). KEYEXPANSION (p. 19) produces an expanded key consisting of the required 44 words. In the following, the key $K = (k_0, k_1, k_2, k_3)$, where the k_i are 4-byte words, and the expanded key is denoted by the word-vector $(w_0, w_1, w_2, \ldots, w_{44})$.

(1) for $i \in \{0, 1, 2, 3\}$, $w_i = k_i$
(2) for $i \in \{4, \ldots, 44\}$:

$$w_i = w_{i-4} \oplus \begin{cases} \textsc{SubWord}(\textsc{RotWord}(w_{i-1})) \oplus \textsc{Rcon}_{i/4} & \text{if } 4 \mid i \\ w_{i-1} & \text{otherwise} \end{cases}$$

The components of KEYEXPANSION are:

- ROTWORD is a one-byte circular left shift on a word.
- SUBWORD performs a byte substitution (using the S-box SUBBYTES on each byte of it's input word).
- RCON is a table of round constants (RCON$_j$ is used in round j). Each is a word with the three rightmost bytes equal to 0.

KEYEXPANSION is similar for 192 and 256-bit keys.

2.5. **Decryption.** To decrypt, perform cipher in reverse order, using inverses of components and the reverse of the key schedule:

(1) ADDROUNDKEY with round key N_r
(2) For rounds $N_r - 1$ to 1 :
  - INVSHIFTROWS
  - INVSUBBYTES
  - ADDROUNDKEY
  - INVMIXCOLUMNS
(3) For round 1 :
  - INVSHIFTROWS
  - INVSUBBYTES
  - ADDROUNDKEY using round key 1

Note: The straightforward inverse cipher has a different sequence of transformations in the rounds. It is possible to reorganize this so that the sequence is the same as that of encryption.