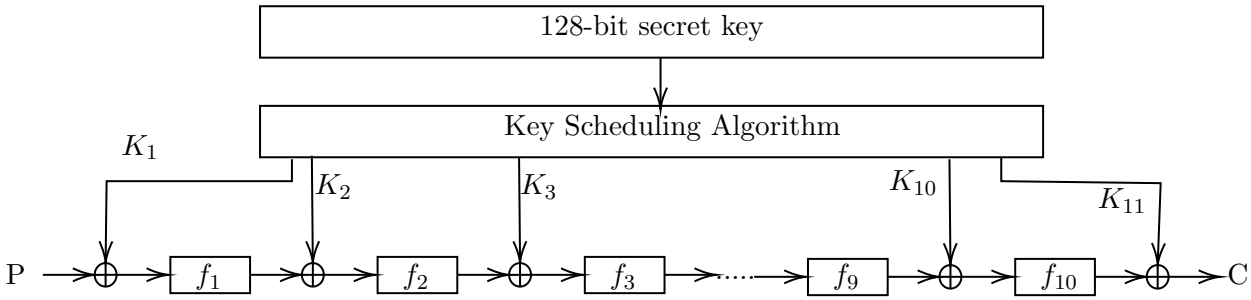
[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy
Scribed by: Sanidhya Kumar (202151138)

Winter 2023-2024
Lecture (Week 5)



AES Structure

1 Components of AES

1.1 Round Function of AES

The Advanced Encryption Standard (AES) employs a round function to iteratively transform plaintext blocks into ciphertext blocks. This round function, denoted as f, comprises various operations applied sequentially, including substitution, permutation, and mixing. Each round function f_i, where i ranges from 1 to 10, maps from a 128-bit input to a 128-bit output:

$$f_i : \{0,1\}^{128} \to \{0,1\}^{128} \quad \text{for} \quad 1 \leq i \leq 10$$

The Subbytes operation is a bijective mapping from 128 bits to 128 bits:

$$\text{Subbytes} : \{0,1\}^{128} \to \{0,1\}^{128}$$

1.1.1 Substitution (SubBytes)

The SubBytes step of AES replaces each byte in the state matrix with another byte using a substitution table called the S-box. This table performs a non-linear byte substitution, which helps enhance the confusion property of AES.

For a byte at row i and column j of the state matrix S, denoted as $S_{i,j}$, the substitution is calculated as:

$$\text{SubBytes}(S)_{i,j} = \text{S-box}(S_{i,j})$$

The input S to the SubBytes function is a 128-bit binary input. A 4×4 matrix can be constructed from this input by arranging the bits in a specific manner.

$$S \rightarrow \begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix}$$

where S_{ij} is a byte (8-bits). Consider the 128-bit plaintext of 128-bit. The plaintext again can be written as a 4×4 matrix in the following way. Keep in mind, the ordering of the plaintext bytes.

$$P = P_0P_1P_2....P_{15}, \text{ length of each } P_i \text{ is 8-bit}$$

$$P \rightarrow \begin{bmatrix} P_0 & P_4 & P_8 & P_{12} \\ P_1 & P_5 & P_9 & P_{13} \\ P_2 & P_6 & P_{10} & P_{14} \\ P_3 & P_7 & P_{11} & P_{15} \end{bmatrix}$$

Similarly, the first round key K_1 can also be written as a matrix.

$$K_1 = K_0K_1K_2....K_{15}, \text{ length of each } K_i \text{ is 8-bit}$$

$$K_1 \rightarrow \begin{bmatrix} K_0 & K_4 & K_8 & K_{12} \\ K_1 & K_5 & K_9 & K_{13} \\ K_2 & K_6 & K_{10} & K_{14} \\ K_3 & K_7 & K_{11} & K_{15} \end{bmatrix}$$

For AES-128, we first xor the plaintext with the first round key K_1. The output is then passed to first round function, wherein, it is first passed to subbytes function.

$$S = (S_{ij})_{4\times4} = P \oplus K_1$$

The output after the subbyte function is performed on S is S'.

$$S' = Subbytes(S)$$

We will see an overview of the subbyte function. Below are the steps involved:

1. A constant C = $C_7C_6C_5C_4C_3C_2C_1C_0 = (01100011) = (63)_{16}$ is declared.
2. Substitution box $\mathbb{S}$ maps every element of the S matrix from 8-bit to 8-bit. This $\mathbb{S}$ box is applied to every element of S matrix, i.e., S_{ij}. Therefore, it is an 8-bit to 8-bit mapping. Also, $\mathbb{S}(0) = 0$ is taken as a rule (fixed for AES).
3. Suppose $\mathbb{S}(S_{ij}) = m_7m_6m_5m_4m_3m_2m_1m_0$.
4. For $i = 0$ to $i = 7$, compute b_i as:

$$b_i = (m_i + m_{(i+4)\%8} + m_{(i+5)\%8} + m_{(i+6)\%8} + m_{(i+7)\%8} + C_i)\%2$$

5. Therefore, the output is $b_7b_6b_5b_4b_3b_2b_1b_0$.
6. $S'_{ij} = (b_7b_6b_5b_4b_3b_2b_1b_0)$

Hence, S'_{ij} is computed for each S_{ij}, and the output matrix is the output of the subbyte function.

$$\begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix} \xrightarrow{Subbyte} \begin{bmatrix} S'_{00} & S'_{01} & S'_{02} & S'_{03} \\ S'_{10} & S'_{11} & S'_{12} & S'_{13} \\ S'_{20} & S'_{21} & S'_{22} & S'_{23} \\ S'_{30} & S'_{31} & S'_{32} & S'_{33} \end{bmatrix}$$

Now, we will discuss the substitution box $\mathbb{S}$ that takes 8-bit input and produces 8-bit output.

$$\mathbb{S} : \{0,1\}^8 \to \{0,1\}^8 \text{ and } \mathbb{S}(0) = 0$$

Let's say input X is given to this $\mathbb{S}$ box and $X \neq 0$. We need to find $Y = \mathbb{S}(X)$.

$$X = a_7a_6a_5a_4a_3a_2a_1a_0, \text{ where } a_i \in \{0,1\}$$

We can construct a polynomial $P(x)$ using the bits of X as coefficients.

$$P(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + ... + a_7 \cdot x^7$$

Clearly, degree of $P(x) \leq 7$. Also, $P(x) \in F_2[x]$. Moreover, since $X \neq 0 \implies P(x) \neq 0$. Another polynomial $g(x) = x^8 + x^4 + x^3 + x + 1$ is fixed for AES. $g(x)$ is a primitive polynomial.

Given that $g(x)$ is a primitive polynomial, it defines a field denoted by $(\mathbb{F}_2[x]/\langle g(x)\rangle, +, *)$, where $+$ and $*$ represent addition and multiplication modulo $g(x)$, respectively. All polynomials within this field have a maximum degree of 7. Consequently, $P(x)$ is a member of this field. Moreover, since $\langle g(x)\rangle$ is primitive, we can ascertain the multiplicative inverse of any polynomial in $(\mathbb{F}_2[x]/\langle g(x)\rangle)$. Hence, we aim to determine the multiplicative inverse of $P(x)$ modulo $g(x)$, denoted as $q(x)$.

$$\begin{aligned} P(x) \cdot q(x) &\equiv 1 mod\ g(x) \\ P(x) \cdot q(x) &\equiv 1 mod (x^8 + x^4 + x^3 + x + 1) \\ P(x) \cdot q(x) - 1 &= h(x) \cdot (x^8 + x^4 + x^3 + x + 1) \\ 1 &= P(x) \cdot q(x) + h(x) \cdot (x^8 + x^4 + x^3 + x + 1) \end{aligned}$$

Therefore, we can find the $q(x)$ with the help of Extended Euclidean Algorithm. Also, $q(x)$ will be a polynomial of degree at most 7.

$$q(x) = r_0 + r_1 \cdot x + r_2 \cdot x^2 + ... + r_7 \cdot x^7, \text{ where } r_i \in \{0,1\}$$

From the coefficients, we can build a 8-bit binary string as $r_7r_6r_5r_4r_3r_2r_1r_0$. This string is the output of the $\mathbb{S}$ box. Therefore,

$$\mathbb{S}(X) = Y = (r_7r_6r_5r_4r_3r_2r_1r_0)$$

Example: Find Subbytes(01010011).

Solution: X = 01010011, therefore $P(x) = x^6 + x^4 + x + 1$. Also, $g(x) = x^8 + x^4 + x^3 + x + 1$. Now, let's perform the Extended Euclidean Algorithm.

$$
\begin{array}{r|l}
x^6+x^4+x+1 & x^8+x^4+x^3+x+1 \quad (x^2+1 \\
\cline{2-2}
 & x^8+x^6+x^3+x^2 \\
\cline{2-2}
 & x^6+x^4+x^2+x+1 \\
 & x^6+x^4+x+1 \\
\cline{2-2}
 & x^2 \quad x^6+x^4+x+1 \quad (x^4+x^2 \\
 & \cline{3-3} x^6 \\
 & \cline{3-3} x^4+x+1 \\
 & x^4 \\
 & \cline{3-3} x+1 \quad x^2 \quad (x+1 \\
 & \cline{3-3} x^2+x \\
 & \cline{3-3} x \\
 & x+1 \\
 & \cline{3-3} 1 \\
 & \cline{3-3}
\end{array}
$$

Now, let's work upside down to find the multiplicative inverse. Therefore, $1 = q(x) \cdot P(x) = h(x) \cdot g(x)$

$$
\begin{aligned}
1 &= 1 \cdot x^2 + (x+1) \cdot (x+1) \\
1 &= x^2 + (x+1) \cdot [(x^6+x^4+x+1) + x^2 \cdot (x^4+x^2)] \\
1 &= (x+1) \cdot (x^6+x^4+x+1) + x^2 \cdot [1 + (x+1) \cdot (x^4+x^2)] \\
1 &= (x+1) \cdot (x^6+x^4+x+1) + x^2 \cdot (x^5+x^4+x^3+x^2+1) \\
1 &= (x+1) \cdot (x^6+x^4+x+1) + (x^5+x^4+x^3+x^2+1) \cdot [(x^8+x^4+x^3+x+1) + (x^6+x^4+x+1) \cdot (x^2+1)] \\
1 &= [(x+1) + (x^2+1) \cdot (x^5+x^4+x^3+x^2+1)] \cdot P(x) + (x^5+x^4+x^3+x^2+1) \cdot g(x) \\
1 &= (x+1+x^7+x^6+x^5+x^4+x^2+x^5+x^4+x^3+x^2+1) \cdot P(x) + (x^5+x^4+x^3+x^2+1) \cdot g(x) \\
1 &= (x^7+x^6+x^3+x) \cdot P(x) + (x^5+x^4+x^3+x^2+1) \cdot g(x)
\end{aligned}
$$

Therefore, Multiplicative Inverse of P(x) is $q(x) = x^7 + x^6 + x^3 + x$. Therefore,

$$\mathbb{S}(01010011) = (11001010) = m_7 m_6 m_5 m_4 m_3 m_2 m_1 m_0$$

Now, let's compute $b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$. The constant C = $c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0 = 01100011$.

	0	1	2	3	4	5	6	7
c	1	1	0	0	0	1	1	0
m	0	1	0	1	0	0	1	1

$$
\begin{gathered}
b_0 = (0+0+0+1+1+1)\%2 = 1 \\
b_1 = (1+0+1+1+0+1)\%2 = 0 \\
b_2 = 1, b_3 = 1, b_4 = 0 \\
b_5 = 1, b_6 = 1, b_7 = 1
\end{gathered}
$$

$$\therefore subbytes(01010011) = b_7b_6b_5b_4b_3b_2b_1b_0 = (11101101)$$
$$subbytes(53) = (ED)$$

The value returned by the *subbyte* function is a hexadecimal number. The first four digits give the row number, the next four digits give the column numbers, and together they highlight the cell in a 16x16 table, which contains the required encrypted characters.

$$\text{Input} = \text{XY}$$

$$\text{Subbyte(Input)} \rightarrow \text{element present in row number X and column number Y}$$

The lookup table is given in the diagram below -

Input	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
01	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
02	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
03	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
04	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
05	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
06	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
07	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
08	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
09	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
0A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
0B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
0C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
0D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
0E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
0F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

1.1.2 Permutation (ShiftRows)

In the ShiftRows step, bytes in each row of the state matrix are cyclically shifted to the left. This permutation operation provides diffusion, spreading the influence of each byte throughout the state matrix.

$$\text{ShiftRows}(S) = \begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,1} & S_{1,2} & S_{1,3} & S_{1,0} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,3} & S_{3,0} & S_{3,1} & S_{3,2} \end{pmatrix}$$

Shift Row function is a mapping from 128-bit to 128-bit. It takes a 4×4 matrix as input (the output of Subbyte function). It performs left circular shift on the elements of i^{th} row by i positions, where row index begins from 0.

$$\text{Shift Rows: } \{0,1\}^{128} \rightarrow \{0,1\}^{128}$$

$$\begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix} \xrightarrow{ShiftRows} \begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{11} & S_{12} & S_{13} & S_{10} \\ S_{22} & S_{23} & S_{20} & S_{21} \\ S_{33} & S_{30} & S_{31} & S_{32} \end{bmatrix}$$

1.1.3 Mixing (MixColumns)

MixColumns is a linear transformation that operates on each column of the state matrix independently. It involves multiplying each column by a fixed matrix and then applying modular polynomial arithmetic. This mixing operation increases the diffusion further.

$$\text{MixColumns}(S) = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{pmatrix}$$

Mix columns, again, is a mapping from 128-bit to 128-bit. It also takes a 4×4 matrix as input (the output of Shift Rows function).

$$\text{Mix Columns: } \{0,1\}^{128} \rightarrow \{0,1\}^{128}$$

$$(S_{ij})_{4\times4} \xrightarrow{MixColumns} (S'_{ij})_{4\times4}$$

Consider the column $c \in 0, 1, 2, 3$ of matrix S.

$$\text{column} = \begin{bmatrix} S_{0c} \\ S_{1c} \\ S_{2c} \\ S_{3c} \end{bmatrix}$$

The Mix Columns function is defined as follows. For i = 0 to i = 3, let t_i be the polynomial constructed from S_{ic}. Define four polynomials as:

$$\begin{aligned} u_0 &= [(x * t_0) + (x+1) * t_1 + t_2 + t_3] \bmod (x^8 + x^4 + x^3 + x + 1) \\ u_1 &= [t_0 + (x * t_1) + (x+1) * t_2 + t_3] \bmod (x^8 + x^4 + x^3 + x + 1) \\ u_2 &= [t_0 + t_1 + (x * t_2) + (x+1) * t_3] \bmod (x^8 + x^4 + x^3 + x + 1) \\ u_3 &= [(x+1) * t_0 + t_1 + t_2 + (x * t_3)] \bmod (x^8 + x^4 + x^3 + x + 1) \end{aligned}$$

Now, S'_{ij} is the binary 8-bits constructed using u_i. Therefore,

$$\begin{bmatrix} S_{0c} \\ S_{1c} \\ S_{2c} \\ S_{3c} \end{bmatrix} \xrightarrow{MixColumns} \begin{bmatrix} S'_{0c} \\ S'_{1c} \\ S'_{2c} \\ S'_{3c} \end{bmatrix}$$

Applying Mix Columns to each columns, will give us the entire $(S'_{ij})_{4\times4}$ matrix. Therefore, Mix Column can be defined as a matrix multiplication as:

$$(S'_{ij})_{4\times4} = \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} \times \begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix} \bmod (x^8 + x^4 + x^3 + x + 1)$$

The polynomial $(x^8 + x^4 + x^3 + x + 1)$ is a primitive polynomial, hence, it is possible to construct the inverse of the Mix Columns function.

1.1.4 Combining everything together!

The round function F is a composition of these three operations:

$$f(S) = \text{MixColumns}(\text{ShiftRows}(\text{SubBytes}(S)))$$

For the first 9 round functions:

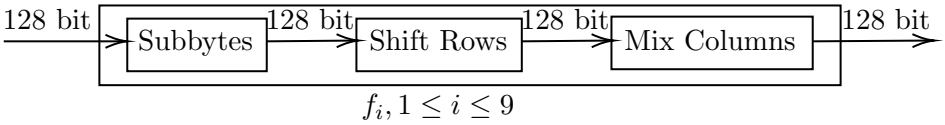
$$f_i(X) = MixColumns(ShiftRows(Subbytes(X))) \ \forall \ 1 \leq i \leq 9$$

However, the last round function f_{10} is based on subbytes and shift rows only. Therefore,

$$f_{10}(X) = ShiftRows(Subbytes(X))$$

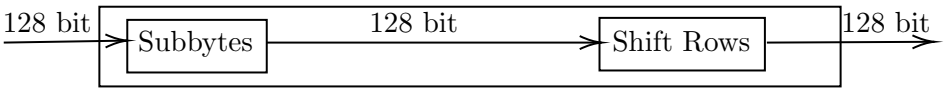
Each Subbyte, Shift Rows and Mix Columns is a bijection from 128-bit to 128-bit. This means each of them has existing inverse. Therefore, given $y = f_i(X)$, to get X, apply inverse of mix columns, then apply inverse of Shift Rows and then apply inverse of Subbytes.

For the first 9 rounds $f_1, f_2, ..., f_9$:



$f_i, 1 \leq i \leq 9$

For the last round function f_{10}:



f_{10}

1.2 Key Scheduling Algorithm of AES-128

The AES-128 key scheduling algorithm operates on a 128-bit encryption key, producing 11 distinct round keys, each consisting of 128 bits. The key is represented as $(key[0], key[1], ..., key[15])$, where each $key[i]$ is a single byte. This process involves generating 44 words, each 32 bits long, denoted as $w[0], w[1], ..., w[43]$. Notably, $\frac{32\times44}{128} = 11$, providing the necessary material for creating 11 round keys from these 44 words.

The AES-128 key scheduling algorithm comprises two essential functions:

1. **ROTWORD($B_0B_1B_2B_3$):** This function takes a word as input and performs a left circular shift of its constituent bytes, equivalent to a rotation by 8 bits. The input word, represented by B_0, B_1, B_2, B_3, undergoes transformation, resulting in the output: ROTWORD($B_0B_1B_2B_3$) = $B_1B_2B_3B_0$.
2. **SUBWORD($B_0B_1B_2B_3$):** This function operates on a word, initiating the Subbyte operation, which is crucial to the round function of AES, on each of its constituent bytes B_0, B_1, B_2, B_3. Consequently, the output of the SUBWORD function takes the form: SUBWORD($B_0B_1B_2B_3$) = $B'_0B'_1B'_2B'_3$, where B'_i = Subbytes(B_i) for all $i \in \{0, 1, 2, 3\}$.

In the AES-128 key scheduling algorithm, ten round constants are employed, each represented as a 32-bit word in hexadecimal format.

In the AES-128 key scheduling algorithm, ten round constants are utilized, each represented as a 32-bit word in hexadecimal format:

$$\begin{aligned} RCON[1] &= 0x01000000 \\ RCON[2] &= 0x02000000 \\ RCON[3] &= 0x04000000 \\ RCON[4] &= 0x08000000 \\ RCON[5] &= 0x10000000 \\ RCON[6] &= 0x20000000 \\ RCON[7] &= 0x40000000 \\ RCON[8] &= 0x80000000 \\ RCON[9] &= 0x1b000000 \\ RCON[10] &= 0x36000000 \end{aligned}$$

These round constants play a crucial role in the key expansion process, contributing to the generation of 44 words, each 32 bits in length. The remaining portion of the algorithm is as follows:

for $i \leftarrow 0$ *to* 3 **do**
end
$w_i \leftarrow key[4i]||key[4i+1]||key[4i+2]||key[4i+3]$;
for $i \leftarrow 4$ *to* 43 **do**
end
temp = w[i-1]; **if** $i\%4 == 0$ **then**
end
temp = SUBWORD(ROTWORD(temp)) $\oplus$ RCON[i/4]; $w[i] = w[i-4] \oplus$ temp;

The resulting 44 words are then utilized to derive the round keys:

$$\begin{aligned} K_1 &= w[0]||w[1]||w[2]||w[3] \\ K_2 &= w[4]||w[5]||w[6]||w[7] \\ &\cdots \\ &\cdots \\ K_{11} &= w[40]||w[41]||w[42]||w[43] \end{aligned}$$

During decryption, the inverse of the key scheduling algorithm is unnecessary, as the round keys are XORed only, ensuring the same round keys can be utilized for encryption and decryption.

1.3 Decryption of AES

In AES decryption, the inverse of the round function is crucial. For rounds 1 to 9, the inverse function is defined as follows:

$$f_i^{-1} = \text{Subbyte}^{-1}(\text{ShiftRows}^{-1}(\text{MixColumns}^{-1}(S))) \text{ for } i \in \{1, 2, \ldots, 9\}$$

For the 10th round function, the inverse is slightly different:

$$f_{10}^{-1} = \text{Subbyte}^{-1}(\text{ShiftRows}^{-1}(S))$$

Inverse Subbyte Function:

The Subbyte function maps an 8-bit input to another 8-bit output using a lookup matrix. To invert this process, the input is divided into two halves to obtain the original value from the lookup matrix.

$$\text{Subbyte}^{-1}(B) = \text{InverseSubbyte}(B) = \text{InverseSubbyte}(X'||Y') = X||Y = A$$

Inverse Shift Row Function:

To reverse the Shift Row operation, the rows are right-circular shifted by the same number of positions they were left-shifted during encryption.

Inverse Mix Column Function:

The Mix Column function is essentially a matrix multiplication, and its inverse can be obtained by multiplying the result with the inverse of the mixing matrix.

$$\text{MixColumns}^{-1}(S) = S' \cdot M^{-1} = S$$

These inverse functions are essential for decrypting data encrypted with AES.

2 Mode of Operation

When employing AES for encryption, data exceeding the 128-bit block size necessitates specific modes of operation. These modes extend the capability to encrypt larger datasets efficiently. Notable modes include:

1. Electronic CodeBook Mode (ECB)
2. Cipher FeedBack Mode (CFB)
3. Cipher Block Chaining Mode (CBC)
4. Output FeedBack Mode (OFB)
5. Counter Mode

6. Count with Cipher Block Chaining Mode (CCM)

In this discussion, focus will be placed on Electronic CodeBook (ECB) and Cipher Block Chaining (CBC) modes.

Consider a scenario where AES encryption is applied to 256-bit data. Initially, the data is divided into 128-bit blocks, with subsequent encryption applied to each block individually. Concatenating the resulting ciphertext yields the encrypted form of the entire 256-bit data. However, when applied to larger data sets, such as a 200kB image, ECB mode exhibits a notable drawback. Identical components within the data, like corresponding features in an image, lead to identical ciphertext segments. Consequently, patterns in the plaintext become discernible from the ciphertext, compromising security.

2.1 Electronic CodeBook Mode

In Electronic CodeBook (ECB) mode, the plaintext undergoes segmentation into continuous blocks of l-bit size, with each block encrypted independently. Concatenating the resulting ciphertext blocks in the same order yields the final ciphertext.

$$M = m_0||m_1||.....||m_t \text{ (plaintext)}$$
$$\text{len}(m_i) = l\text{-bit (each } m_i \text{ is a block, for AES, } l = 128)$$

Encryption:

$$C = C_0||C_1||.....||C_t$$
$$C_i = \text{Enc}(m_i, K) \quad \forall \quad i \in \{0, 1, ..., t\}$$

Decryption:

$$M = m_0||m_1||.....||m_t$$
$$m_i = \text{Dec}(C_i, K) \quad \forall \quad i \in \{0, 1, ..., t\}$$

While ECB mode facilitates parallel encryption of multiple blocks, its susceptibility to information leakage arises when identical plaintext blocks result in identical ciphertext blocks, i.e., $m_i = m_j$ implies $C_i = C_j$.

2.2 Cipher Block Chaining Mode

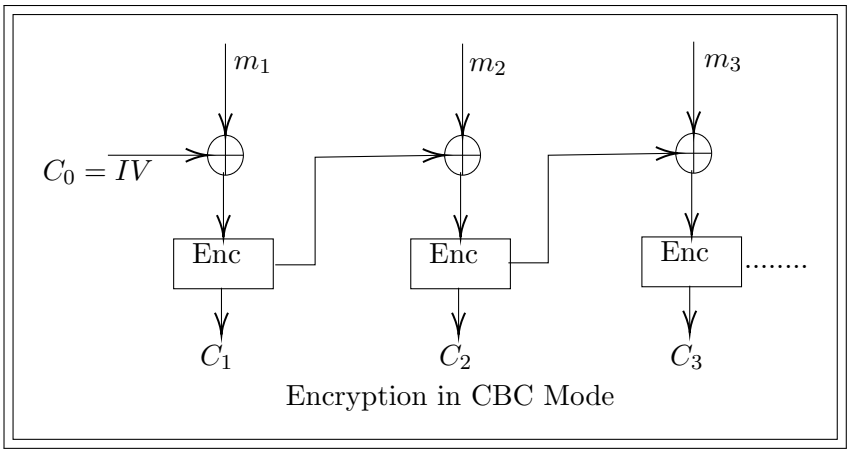
Cipher Block Chaining (CBC) mode is a widely used method in cryptography. It relies on an l-bit public block called the Initialization Vector (IV), which is crucial for the encryption process.

Encryption:

$$M = m_1||m_2||.....||m_t \text{ (plaintext)}$$
$$\text{len}(m_i) = l\text{-bit (each } m_i \text{ is a block, for AES, } l = 128)$$

In CBC mode, the ciphertext comprises (n+1) blocks for a plaintext of n blocks. The encryption process unfolds as follows:

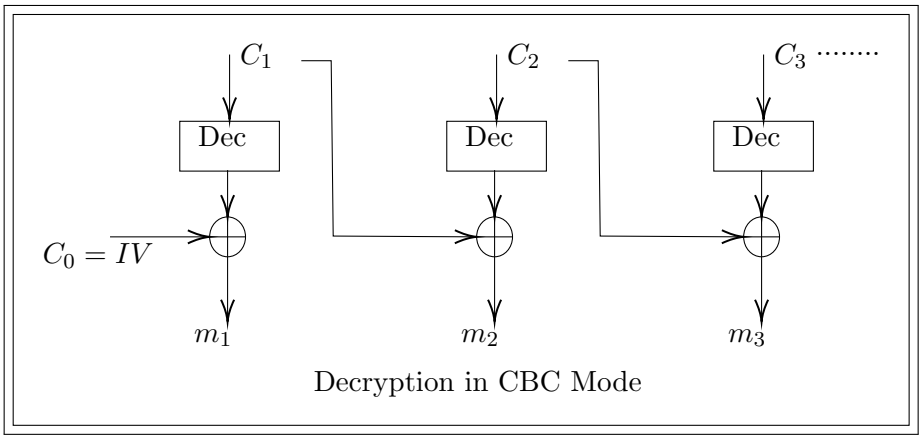
$$C_0 = IV$$
$$C_i = \text{Enc}(C_{i-1} \oplus m_i, K) \quad \forall \quad i \in \{1, 2, 3...., t\}$$
$$C = C_0||C_1||.....||C_t$$



Encryption in CBC Mode

Decryption:

$$m_i = Dec(C_i, K) \oplus C_{i-1} \ \forall \ i \in \{1, 2..., t\}$$
where $C_0 = IV$
$$M = m_1||m_2||.....||m_t$$



Decryption in CBC Mode

3 Stream Cipher

Stream ciphers operate on the level of individual bits, encrypting data bit by bit. Let $M = m_0 m_1 ... m_l$, where $m_i \in \{0, 1\}$ represents each bit of the plaintext message. The encryption process involves combining each plaintext bit m_i with a corresponding keystream bit k_i generated by the stream cipher algorithm, resulting in the ciphertext C:

$$C = M \oplus K = (m_0 \oplus k_0)(m_1 \oplus k_1)...(m_l \oplus k_l)$$

Where $\oplus$ denotes the bitwise XOR operation.

Encryption: $C_i = m_i \oplus K_i$

Decryption: $m_i = C_i \oplus K_i$

3.1 Shannon's Notion of Perfect Secrecy

Shannon's Theorem of Perfect Secrecy stands as a fundamental principle within cryptography, rigorously defining an algorithm's perfect security as one where the ciphertext divulges no additional information about the plaintext, irrespective of an adversary's computational capabilities.

Mathematically, Shannon's theorem can be expressed as follows:

The probability of a specific plaintext message m_1 occurring, denoted as $Pr[M = m_1]$, remains unchanged even when conditioned on a particular ciphertext CH_1, denoted as $Pr[M = m_1|C = CH_1]$. This equality signifies that the ciphertext does not offer any extra insight into the likelihood of m_1.

For illustration, let's consider binary variables m and k, where m (belonging to the set $\{0, 1\}$) represents the plaintext bit and k (also from the set $\{0, 1\}$) denotes the corresponding key bit. Assuming the probabilities of m and k being 0 are P and $\frac{1}{2}$ respectively, and similarly for 1, we can compute the probabilities of the resulting ciphertext C as follows:

$$\begin{aligned} Pr[C = 0] &= Pr[M = 0 \wedge K = 0] + Pr[M = 1 \wedge K = 1] \\ &= P \times \frac{1}{2} + (1 - P) \times \frac{1}{2} \\ &= \frac{1}{2} \\ Pr[C = 1] &= Pr[M = 0 \wedge K = 1] + Pr[M = 1 \wedge K = 0] \\ &= P \times \frac{1}{2} + (1 - P) \times \frac{1}{2} \\ &= \frac{1}{2} \end{aligned}$$

This demonstrates that the resulting ciphertext C is uniformly distributed. Thus, even if the plaintext displays bias, the ciphertext remains effectively randomized, thereby preserving perfect secrecy.

3.2 Conditions for Perfect Secrecy

In order for a stream cipher to achieve perfect secrecy, it must fulfill the following criteria:

1. **Unique Keys:** Each message must be encrypted using a different key. Reusing the same key for multiple messages can lead to potential information leakage, as patterns in the ciphertext may reveal aspects of the plaintext.

2. **Key Length vs. Message Length:** The length of the key ($|K|$) should be equal to or greater than the length of the message ($|M|$). If the key length is shorter than the message length ($|K| < |M|$), padding the key with repeated bits to match the message length can introduce vulnerabilities, potentially exposing information about the plaintext.

Adhering to these conditions ensures that the stream cipher maintains perfect secrecy, preventing any inference about the plaintext from the ciphertext, even in the presence of adversaries with significant computational resources.

For example, the Vernam cipher, also known as the **One-Time Pad (OTP)**, achieves perfect secrecy by using each bit of the key only once. However, OTP is impractical due to the challenges of securely sharing keys that are as long as the messages themselves.