[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023

Scribed by: Chitranshi Srivastava (202051055) — Lecture 10 (Week 6)

In previous week, we have studied the mathematics required for understanding the AES algorithm - Generators, Subgroup, Lagrange's Theorem, Ring, Field and Field Extension, Polynomial Ring, Irreducible Polynomials and Primitivie Polynomials.

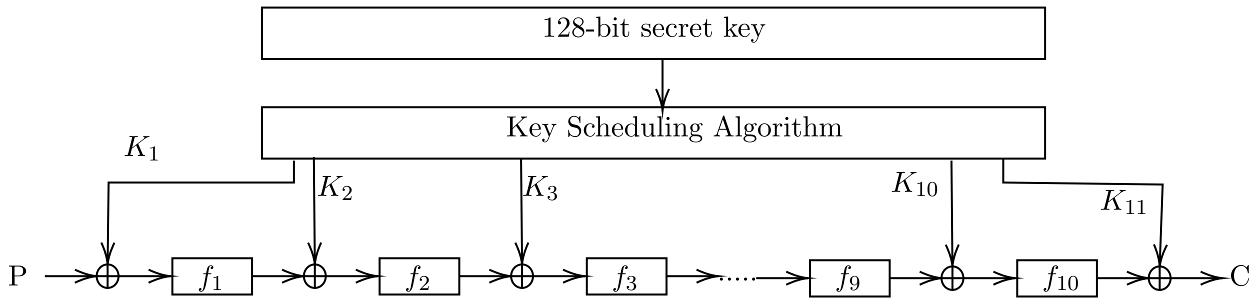
1 Advanced Encryption Standard (AES)

When the design of DES was made public, it was immediately broken. After that, NIST called for a competition known as Advanced Encryption Standard. There were a lot of cryptographers around the world who submitted their designs along with the implementation. One of the submission in the competition was *Rijndael.* It was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen. AES is unbreakable till date.

Advanced encryption Standard is an iterated block cipher and is based on Substitution Permutation Network(SPN). There are three different variants of AES:

1. AES-128
   - Block Size 128-bit
   - Number of Rounds = 10
   - Secret Key Size 128-bit
2. AES-192
   - Block Size 128-bit
   - Number of Rounds = 12
   - Secret Key Size 192-bit
3. AES-256
   - Block Size 128-bit
   - Number of Rounds = 14
   - Secret Key Size 256-bit

2 AES 128



The diagram above depicts the encryption process.
The Key Scheduling Algorithm takes 128-bit key as input and generates 11 round keys of 128-bit each. The plaintext P is xored with K_1(first round key). This value is input to f_1(first round function) which produces 128-bit output. This output is cored with K_2(second round key) and the output is passed to f_2(second round function). Again, f_2 produces 128-bit output and this will continue till f_{10}. Finally, output of f_{10} will be xored with K_{11}(last round key) and this output will be our ciphertext.

Given the ciphertext, decryption process will begin with xoring the ciphertext with K_{11}(last round key) and this output will be input to inverse of f_{10} function. Inverse of f_{10} function will generate a 128-bit output which will be xored with K_{10}, and output will be passed to inverse of f_9 function, and so on it will be continued till inverse of f_1 function. The output will be xored with K_1(first round key) to get the plaintext.

Note: The decryption process described above clearly states that the round functions $f_1, f_2, ..., f_{10}$ must be invertible. Hence, this structure is different from Feistel Network where the invertibility of round functions does not matter.
Now, we need to understand the following things:

- Round Functions
- Key Scheduling Algorithm

2.1 Round Functions of AES-128

The round functions $f_1, f_2, ..., f_{10}$ are a mapping from 128-bit to 128-bit.

$$f_i : \{0,1\}^{128} \rightarrow \{0,1\}^{128} \ \forall \ 1 \leq i \leq 10$$

The first 9 round functions $f_1, f_2, ..., f_9$ are same and are different from the last round function f_{10}. This is true for other variants of AES too. The last round function is different from the other round functions. All the round functions are same except the last one.

The first 9 round functions $f_1, f_2, ..., f_9$ are based on the following three functions:

- Subbytes
- Shift Rows
- Mix Columns

And for the last round function, we have the following steps:

- Subbytes
- Shift Rows

For the first 9 round functions:

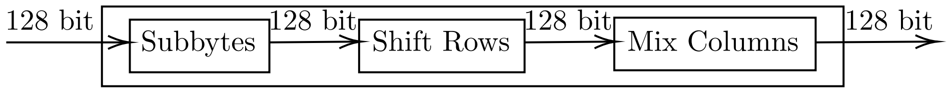
$$f_i(X) = MixColumns(ShiftRows(Subbytes(X))) \ \forall \ 1 \leq i \leq 9$$

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

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

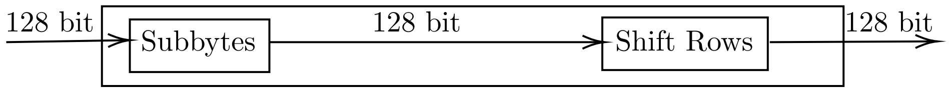
Each Subbyte, Shift Rows and Mix Columns is a bijection from 128-bit to 128-bit. This means each of them has existing inverse. Therefore, given $y = f_i(X)$, to get X, apply inverse of mix columns, then apply inverse of Shift Rows and then apply inverse of Subbytes.

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}

2.1.1 Subbytes

Subbytes is a bijective mapping from 128-bit to 128-bit.

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

The input S to subbytes function is a 128-bit binary input. A 4×4 matrix can be constructed from the input in the following way:

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

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.

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

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

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

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

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

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'.

$$S' = Subbytes(S)$$

Now, let us look at how the subbyte function works. The following steps are followed in order to get the output from subbyte function.

1. Declare a constant C = $C_7C_6C_5C_4C_3C_2C_1C_0$ = (01100011) = $(63)_{16}$
2. There is a substitution box $\mathbb{S}$ from 8-bit to 8-bit. This $\mathbb{S}$ box is applied to every element of S matrix, i.e, S_{ij}. Also, $\mathbb{S}(0) = 0$ is always true.
3. Now, let's say $\mathbb{S}(S_{ij}) = m_7m_6m_5m_4m_3m_2m_1m_0$.
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 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.

Since, $g(x)$ is primitive, $(F_2[x]/\langle g(x)\rangle, +, *)$ is a field where $+$ is addition modulo $g(x)$ and $*$ is multiplication modulo $g(x)$. All the polynomials in the above field will have at most degree 7. Therefore, $P(x)$ belongs to above field. Also, since $\langle g(x)\rangle$ is primitive, we can find multiplicative inverse of any polynomial in $(F_2[x]/\langle g(x)\rangle)$. Therefore, we will find the multiplicative inverse of $P(x)$ under modulo $g(x)$. Let's say the multiplicative inverse of $P(x)$ under modulo $g(x)$ be $q(x)$. Therefore,

$$P(x) \cdot q(x) \equiv 1 mod\ g(x)$$
$$P(x) \cdot q(x) \equiv 1 mod(x^8 + x^4 + x^3 + x + 1)$$
$$P(x) \cdot q(x) - 1 = h(x) \cdot (x^8 + x^4 + x^3 + x + 1)$$
$$1 = P(x) \cdot q(x) + h(x) \cdot (x^8 + x^4 + x^3 + x + 1)$$

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.

$x^6 + x^4 + x + 1 \,\big)\; x^8 + x^4 + x^3 + x + 1 \;\big(\, x^2 + 1$

$x^8 + x^6 + x^3 + x^2$

$x^6 + x^4 + x^2 + x + 1$

$x^6 + x^4 + x + 1$

$x^2 \,\big)\; x^6 + x^4 + x + 1 \;\big(\, x^4 + x^2$

x^6

$x^4 + x + 1$

x^4

$x + 1 \,\big)\; x^2 \;\big(\, x + 1$

$x^2 + x$

---

x

$x + 1$

1

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

$1 = 1 \cdot x^2 + (x + 1) \cdot (x + 1)$

$1 = x^2 + (x + 1) \cdot [(x^6 + x^4 + x + 1) + x^2 \cdot (x^4 + x^2)]$

$1 = (x + 1) \cdot (x^6 + x^4 + x + 1) + x^2 \cdot [1 + (x + 1) \cdot (x^4 + x^2)]$

$1 = (x + 1) \cdot (x^6 + x^4 + x + 1) + x^2 \cdot (x^5 + x^4 + x^3 + x^2 + 1)$

$1 = (x+1) \cdot (x^6+x^4+x+1) + (x^5+x^4+x^3+x^2+1) \cdot [(x^8+x^4+x^3+x+1) + (x^6+x^4+x+1) \cdot (x^2+1)]$

$1 = [(x + 1) + (x^2 + 1) \cdot (x^5 + x^4 + x^3 + x^2 + 1)] \cdot P(x) + (x^5 + x^4 + x^3 + x^2 + 1) \cdot g(x)$

$1 = (x + 1 + x^7 + x^6 + x^5 + x^4 + x^2 + x^5 + x^4 + x^3 + x^2 + 1) \cdot P(x) + (x^5 + x^4 + x^3 + x^2 + 1) \cdot g(x)$

$1 = (x^7 + x^6 + x^3 + x) \cdot P(x) + (x^5 + x^4 + x^3 + x^2 + 1) \cdot g(x)$

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

$$\mathbb{S}(01010011) = (11001010) = m_7m_6m_5m_4m_3m_2m_1m_0$$

Now, let's compute $b_7b_6b_5b_4b_3b_2b_1b_0$. The constant C = $c_7c_6c_5c_4c_3c_2c_1c_0 = 01100011$.

	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

$$b_0 = (0+0+0+1+1+1)\%2 = 1$$
$$b_1 = (1+0+1+1+0+1)\%2 = 0$$
$$b_2 = 1, b_3 = 1, b_4 = 0$$
$$b_5 = 1, b_6 = 1, b_7 = 1$$
$$\therefore subbytes(01010011) = b_7b_6b_5b_4b_3b_2b_1b_0 = (11101101)$$
$$subbytes(53) = (ED)$$

The 8-bit to 8-bit mapping can be stores separately in a table of 256 inputs and corresponding 256 outputs with a memory of just $2^8 \times 8$ bits. Hence, this computation is not done. The value is looked up in a $16 \times 16 table$.

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

The lookup table is given in the Handbook of Cryptography book and we can verify the above example using that.