[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2022-2023
Scribed by: Akshay (202051018) — Lecture (Week 7)

1 Shift rows

$$ShiftRows : \{0,1\}^{128} \rightarrow \{0,1\}^{128}$$

$$\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array}\begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{bmatrix} \rightarrow \begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{11} & s_{12} & s_{13} & s_{10} \\ s_{22} & s_{23} & s_{20} & s_{21} \\ s_{33} & s_{30} & s_{31} & s_{32} \end{bmatrix}$$

We left rotate i^{th} row by i unit.

2 Mix Column

$$MixColumn : \{0,1\}^{128} \rightarrow \{0,1\}^{128}$$
$$(s_{ij})_{4\times4} \rightarrow (s'_{ij})_{4\times4}$$

Consider the column $C \in \{0,1,2,3\}$

$$\text{Column} = \begin{pmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{pmatrix}$$

$s_{ic} = (a_7a_6a_5 \text{ .... } a_0)$
$Poly^n = a_0 + a_1x + ... + a_7x^7$

For i=0 to 3:
    t_i = Binary to poly(s_{ic})

$$u_0 = \Big[(x*t_0) + (x+1)*t_1 + t_2 + t_3\Big] mod(x^8+x^4+x^3+x+1)$$
$$u_1 = \Big[(x*t_1) + (x+1)*t_2 + t_3 + t_0\Big] mod(x^8+x^4+x^3+x+1)$$
$$u_2 = \Big[(x*t_2) + (x+1)*t_3 + t_0 + t_1\Big] mod(x^8+x^4+x^3+x+1)$$
$$u_3 = \Big[(x*t_3) + (x+1)*t_0 + t_1 + t_2\Big] mod(x^8+x^4+x^3+x+1)$$

s'_{ic} = Polynomial_to_binary(u_i)

$$\begin{pmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{pmatrix} \rightarrow \begin{pmatrix} s'_{0c} \\ s'_{1c} \\ s'_{2c} \\ s'_{3c} \end{pmatrix}$$

Matrix form:

$$\begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} * \begin{bmatrix} poly(s_{00}) & poly(s_{01}) & poly(s_{02}) & poly(s_{03}) \\ poly(s_{10}) & poly(s_{11}) & poly(s_{12}) & poly(s_{13}) \\ poly(s_{20}) & poly(s_{21}) & poly(s_{22}) & poly(s_{23}) \\ poly(s_{30}) & poly(s_{31}) & poly(s_{32}) & poly(s_{33}) \end{bmatrix} mod(x^8+x^4+x^3+x+1)$$

$$= \begin{bmatrix} (u_0)_{c=0} & (u_0)_{c=1} & (u_0)_{c=2} & (u_0)_{c=3} \\ (u_1)_{c=0} & (u_1)_{c=1} & (u_1)_{c=2} & (u_1)_{c=3} \\ (u_2)_{c=0} & (u_2)_{c=1} & (u_2)_{c=2} & (u_2)_{c=3} \\ (u_3)_{c=0} & (u_3)_{c=1} & (u_3)_{c=2} & (u_3)_{c=3} \end{bmatrix}$$

$$\begin{bmatrix} to\_binary(\ (u_0)_{c=0}\ ) & to\_binary(\ (u_0)_{c=1}\ ) & to\_binary(\ (u_0)_{c=2}\ ) & to\_binary(\ (u_0)_{c=3}\ ) \\ to\_binary(\ (u_1)_{c=0}\ ) & to\_binary(\ (u_1)_{c=1}\ ) & to\_binary(\ (u_1)_{c=2}\ ) & to\_binary(\ (u_1)_{c=3}\ ) \\ to\_binary(\ (u_2)_{c=0}\ ) & to\_binary(\ (u_2)_{c=1}\ ) & to\_binary(\ (u_2)_{c=2}\ ) & to\_binary(\ (u_2)_{c=3}\ ) \\ to\_binary(\ (u_3)_{c=0}\ ) & to\_binary(\ (u_3)_{c=1}\ ) & to\_binary(\ (u_3)_{c=2}\ ) & to\_binary(\ (u_3)_{c=3}\ ) \end{bmatrix} = (s'_{ij})_{4\times 4}$$

Hexadecimal representation:
$x = (00000010)_2 = (2)_{16}$
$x+1 = (00000011)_2 = (3)_{16}$
$1 = (00000001)_2 = (1)_{16}$

2.1 Illustration

Find $\begin{bmatrix} s'_{00} \\ s'_{10} \\ s'_{20} \\ s'_{30} \end{bmatrix}$ after doing mix-column operation on $\begin{bmatrix} s_{00} \\ s_{10} \\ s_{20} \\ s_{30} \end{bmatrix}$
where, $s_{00} = 95$, $s_{10} = 65$, $s_{20} = FD$, $s_{30} = F3$

Sol:

c = 0 (given)

So, $$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} * \begin{bmatrix} poly(s_{00}) \\ poly(s_{10}) \\ poly(s_{20}) \\ poly(s_{30}) \end{bmatrix} mod(x^8+x^4+x^3+x+1)$$

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} * \begin{bmatrix} poly(95) \\ poly(65) \\ poly(FD) \\ poly(F3) \end{bmatrix} mod(x^8+x^4+x^3+x+1)$$

$$= \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} * \begin{bmatrix} poly(1001\ 0101) \\ poly(0110\ 0101) \\ poly(1111\ 1101) \\ poly(1111\ 0011) \end{bmatrix} mod(x^8+x^4+x^3+x+1)$$

$$= \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} * \begin{bmatrix} x^7+x^4+x^2+1 \\ x^6+x^5+x^2+1 \\ x^7+x^6+x^5+x^4+x^3+x^2+1 \\ x^7+x^6+x^5+x^4+x^1+1 \end{bmatrix} mod(x^8+x^4+x^3+x+1)$$

$$= \begin{bmatrix} x^8+x^7+x^3+x+1 \\ x^8+x^7+x^5+x^3+x+1 \\ x^4+x^3+x^2+x+1 \\ x^7+x^6+1 \end{bmatrix} mod(x^8+x^4+x^3+x+1)$$

Trick to multiply and sum faster:

1. Write x^8 to x^0 as a header of a table
2. After each individual computations of terms, append 1 to the column of corresponding degree
3. Result will have only those degree terms which got odd count of 1s.

To find remainder (or to solve modulo) replace the highest degree term from the dividend with the part of di

Part of divisor: x^4+x^3+x+1

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} x^8+x^7+x^3+x+1 \\ x^8+x^7+x^5+x^3+x+1 \\ x^4+x^3+x^2+x+1 \\ x^7+x^6+1 \end{bmatrix} mod(x^8+x^4+x^3+x+1)$$

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} (x^4+x^3+x+1)+x^7+x^3+x+1 \\ (x^4+x^3+x+1)+x^7+x^5+x^3+x+1 \\ x^4+x^3+x^2+x+1 \\ x^7+x^6+1 \end{bmatrix}$$

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} x^7+x^4 \\ x^7+x^5+x^4 \\ x^4+x^3+x^2+x+1 \\ x^7+x^6+1 \end{bmatrix}$$

Now,

$$\begin{bmatrix} s'_{00} \\ s'_{10} \\ s'_{20} \\ s'_{30} \end{bmatrix} = \begin{bmatrix} to\_binary(u_0) \\ to\_binay(u_1) \\ to\_binay(u_2) \\ to\_binay(u_3) \end{bmatrix}$$

$$\begin{bmatrix} s'_{00} \\ s'_{10} \\ s'_{20} \\ s'_{30} \end{bmatrix} = \begin{bmatrix} to\_binary(x^7+x^4) \\ to\_binay(x^7+x^5+x^4) \\ to\_binay(x^4+x^3+x^2+x+1) \\ to\_binay(x^7+x^6+1) \end{bmatrix}$$

$$\begin{bmatrix} s'_{00} \\ s'_{10} \\ s'_{20} \\ s'_{30} \end{bmatrix} = \begin{bmatrix} 1001\ 0000 \\ 1011\ 0000 \\ 0001\ 1111 \\ 1100\ 0001 \end{bmatrix}$$

$$\begin{bmatrix} s'_{00} \\ s'_{10} \\ s'_{20} \\ s'_{30} \end{bmatrix} = \begin{bmatrix} 90 \\ B0 \\ 1F \\ C1 \end{bmatrix}$$

We are done with round functions of AES with full clarity!!

3 Key Scheduling in AES

Input : 128 bit key
Output : 11 round keys. Length of each round key is 128 bit.
Key = $(Key[15], ..., Key[0])$ (16 bytes)
We will prepare 44 words which are denoted by w[0], ..., w[43].
Size of each word is 32 bits.
44 words $\implies$ 1408 bits = 11*128 bits $\implies$ 11 round keys

Two utility functions:

- ROTWORD$(B_0, B_1, B_2, B_3) = (B_1, B_2, B_3, B_0)$
- SUBWORD$(B_0, B_1, B_2, B_3) = (B'_0, B'_1, B'_2, B'_3)$
  where, B'_i = SUBBYTES(B_i) $\forall\ i = 0, ..., 3$

Each B_i is 1 byte.

10 round constants (word) : (Represented in hexadecimal)

- RCon[1] = 01000000
- RCon[2] = 02000000
- RCon[3] = 04000000
- RCon[4] = 08000000
- RCon[5] = 10000000
- RCon[6] = 20000000
- RCon[7] = 40000000
- RCon[8] = 80000000
- RCon[9] = 1B000000
- RCon[10]= 36000000

```
for i=0 to 3: (generating first 4 words)
    w[i] = concate(Key[4i], Key[4i+1], Key[4i+2], Key[4i+3])
Each such key[] is of 1 byte.
for i=4 to 43:
    temp = w[i-1]
    if i $\equiv$ 0 mod4
        temp = SUBWORD(ROTWORD(temp)) $\oplus$ RCon[i/4]
    w[i] = w[i-4] $\oplus$ temp
return (w[0], ...., w[43])
```

Round Keys $K_1,\ K_2,\ ....\ ,K_{11}$:
K_1 = w[0] || w[1] || w[2] || w[3]
K_2 = w[4] || w[5] || w[6] || w[7]
.
.
.
.
K_{11} = w[40] || w[41] || w[42] || w[43]

4 For Decryption of AES

4.1 Inverse of Subbyte

Subbyte(A) = B, A = X||Y
X = row number and Y = column number of 16 x 16 subbyte table.
For inverse we need to find A corresponding to a B.
Loop through the whole table to locate B in the table (atmost 256 iteration it will take). Now we can get X and Y from the location.

4.2 Inverse Shift Rows

Right rotate i^th row by i unit.

4.3 MixColumn Inverse

$MixCol(S) = S^{'} = M * S$

Where, M is $\begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix}$

Proven result:
MixCol(MixCol(MixCol(MixCol(S)))) = $M^4 * S = I * S$
$M^4 = I$
So, in inverse function of MixCol we can apply MixCol function thrice recursively.

5 Modes of Operation

5.1 ECB (Electronic Code Block mode)

Encryption algorithms like AES and DES cannot encrypt an input of size beyond some maximum size, at a time. So, we can divide our input (or plaintext) in blocks and apply encryption on each block separately. This is what done under ECB mode. Under ECB each block is encrypted independently. But, is it safe?
Problem occurs when we have 2 or more similar blocks in our input. For e.g., while encrypting an image eyes of a person end up in different blocks then cipher of them will become same. This will reveal information about the original image. Pattern of plaintext get revealed. Although the cryptography algorithm is very strong and secure, the way we are using the algorithm creating vulnerabilities.

5.2 CFB (Cipher Feedback mode)

CFB mode also generates a keystream for use in a stream cipher, but this time the resulting stream cipher is asynchronous. We start with $y_0 = IV$ (an initialization vector) and we produce the keystream element z_i by encrypting the previous ciphertext block. That is,

$$z_i = e_K(y_{i-1}),$$

for all $i \geq 1$. As in OFB mode, we encrypt using the formula

$$y_i = x_i \oplus z_i,$$

for all $i \geq 1$. Again, the encryption function e_K is used for both encryption and decryption in CFB mode.

5.3 CBC (Cipher Block Chaining mode) [Most used]

$$M = m_1||m_2||....||m_t$$

Initialization vector = IV (Public variable)
Enc $\rightarrow$ block size l.
len$(m_i) = l$, len(IV) $= l$

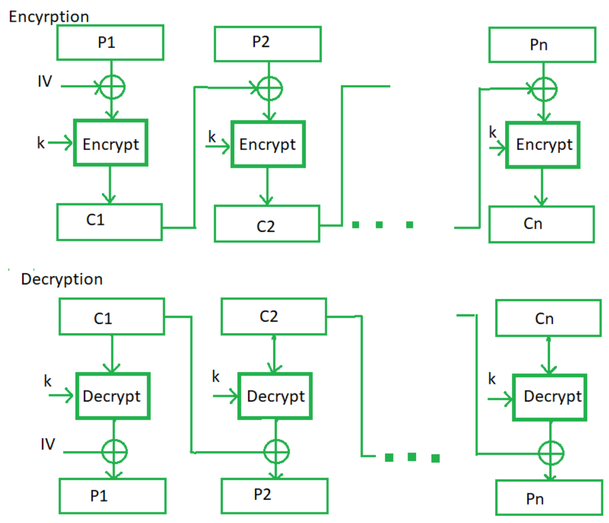
5.3.1 Encryption

$C_0 =$ IV
$C_i = Enc(C_{i-1} \oplus m_i, K)$ $\forall$ i=1, ..., t

Ciphertext:
$C = C_0||C_1||C_2....||C_t$

5.3.2 Decryption

$m_i = Dec(C_i, K) \oplus C_{i-1}$ $\forall$ i=1,....,t



Source: GeeksforGeeks

5.4 OFB (Output Feedback mode)

In OFB mode, a keystream is generated, which is then x-ored with the plaintext. OFB mode is actually a synchronous stream cipher: the keystream is produced by repeatedly encrypting an initialization vector, IV. We define $z_0 = IV$, and then compute the keystream $z_1 z_2 \ldots$ using the rule

$$z_i = e_K(z_{i-1}),$$

for all $i \geq 1$. The plaintext sequence $x_1 x_2 \ldots$ is then encrypted by computing

$$y_i = x_i \oplus z_i,$$

for all $i \geq 1$.

Decryption is straightforward. First, recompute the keystream $z_1 z_2 \ldots$, and then compute

$$x_i = y_i \oplus z_i$$

, for all $i \geq 1$. Note that the encryption function e_K is used for both encryption and decryption in OFB mode.

5.5 Counter mode

Counter mode is similar to OFB mode; the only difference is in how the keystream is constructed. Suppose that the length of a plaintext block is denoted by m. In counter mode, we choose a *counter*, denoted ctr, which is a bitstring of length m. Then we construct a sequence of bitstrings of length m, denoted $T_1, T_2, \ldots$, defined as follows:

$$T_i = ctr + i - 1 mod 2^m$$

for all $i \geq 1$. Then we encrypt the plaintext blocks $x_1, x_2, \ldots$ by computing

$$y_i = xi \oplus e_K(T_i),$$

for all $i \geq 1$. Observe that the keystream in counter mode is obtained by encrypting the sequence of counters using the key K.

As in the case of OFB mode, the keystream in counter mode can be constructed independently of the plaintext. However, in counter mode, there is no need to iteratively compute a sequence of encryptions; each keystream element $e_K(T_i)$ can be computed independently of any other keystream element. (In contrast, OFB mode requires one to compute z_{i-1} prior to computing z_i.) This feature of counter mode permits very efficient implementations in software or hardware by exploiting opportunities for parallelism.

5.6 CCM (Counter with Cipher block Chaining Mode)

Basically, CCM mode combines the use of counter mode (for encryption) with CBC-mode (for authentication).

6 Hash Function

$$h : A \rightarrow B$$

h(X) = Y

- If X is altered to $X^{'}$ then h($X^{'}$) will be completely different from h
- Given Y it is practically infeasible to find X s.t. h(X) = Y
- Given X and Y = h(X) it is practically infeasible to find $X^{'}$ s.t. h(X) = h($X^{'}$)

Consider the conversation b/w two person (Savitri and Max) :

$$\begin{tikzcd}
\textbf{Savitri} & \textbf{Max} \\
X{=}E(M,K) \arrow[r, "X"] & X_1{=}\ \mathrm{Dec}(\tilde{X},\ K) \\
S_1{=}h(M,K) \arrow[r, "S_1"] & S_2{=}\ h(X_1,\ K)
\end{tikzcd}$$

Where,
X is the correct cipher (sent by Savitri to Max)
$\tilde{X}$ is the cipher received by the Max (which may or may not be altered/sent by the Savitri)
X_1 is the message which Max got after decryption
S_1 is the value Savitri got by applying hash function on her message. It is also sent to the Max.
h is some hash function
S_2 is the value Max got by applying same hash function on the X_1

If S_2 is found not to be equal to S_1, then it means X_1 is not equal to X $\implies$ the message which Max received is not sent by the Savitri. (Authentication)

6.1 Defining

A hash family is a four tuple (P, S, K, H) where the following conditions are satisfied.

1. P is the set of all possible messages
2. S is the set of all possible message digests or authentication tags
3. K is the key space
4. H is the set of all hash functions

5. For each $K_1 \in$ K, there is a hash function $h_{K_1} \in$ H s.t.

$$h_{K_1} : P \to S$$

Here, $|P| \geq |S|$
more interestingly, $|P| \geq 2 * |S|$

- If key is involved in the computation of hashed value then that hash function is known as **Keyed hash function**.
- If key is not required to compute the hashed value then that hash function is known as **Unkeyed hash function**.

In general, we will talk about Unkeyed hash function.

6.2 Problem 1

Given $y \in S$, find $x \in P$ such that $h(x) = y$
This problem is known as **preimage finding problem**.
For an hash function h if you cannot find preimage in a feasible time then h is known as **preimage resistant hash function**.
Finding preimage is computationally hard for preimage resistant hash function.

6.3 Problem 2

Given $x \in P$ find $x^{'} \in P$ s.t. $x^{'} \neq x$ and $h(x^{'}) = h(x)$.
The problem is known as **Second pre-image finding problem**.
If finding second preimage is computationally hard for h then h is known as **Second preimage resistant hash function**.

6.4 Problem 3

Find $x, x^{'} \in P$ s.t. $x \neq x^{'}$ and $h(x) = h(x^{'})$
This problem is known as **Collision finding problem**.
For an hash function h if finding collision is computationally hard then h is known as collision resistant hash function.

6.5 Ideal Hash function

h will be called ideal hash function if given $x \in P$ to find h(x) either you have to apply h on x or you have to look into the table corresponding to h (hash table).

6.6 Pre-image finding Algo

$$h : X \to Y; \qquad |Y| = M$$

Given $y \in Y$ find $x \in X$ s.t. $h(x) = y$

```
Choose any $X_0 \subseteq X$ s.t. $|X_0| = Q$
for each $x \in X_0$ :
    compute $y_x = h(x)$
    if $y_x = y$
```

```
        return $x$
```

$X_0 = \{x_1, x_2, ..., x_Q\}$
E_i : event $h(x_i) = y$; $1 \le i \le Q$
$Pr[E_i] = \frac{1}{M}$

Reasoning: Favourable case(s): h(x) = given y; All cases: h(x) can take any value out of m values in set Y.

$Pr[E_i^c] = 1 - \frac{1}{M}$

$Pr[success] = Pr[E_1 \cup E_2 \cup E_3 \cup .... \cup E_Q] = 1 - Pr[E_1^c \cap E_2^c \cap E_3^c \cap .... \cap E_Q^c]$

Failure of one event is independent of the other events, so

$$\begin{aligned} Pr[success] &= 1 - \Pi_{i=1}^{Q} Pr[E_i^c] = 1 - \left(1 - \frac{1}{M}\right)^Q \\ &= 1 - \left[1 - \binom{Q}{1}\frac{1}{M} + \binom{Q}{2}\frac{1}{M^2} - \ \ .... \right] \\ &\approx \left[1 - \binom{Q}{1}\frac{1}{M}\right] \\ &= \frac{Q}{M} \end{aligned}$$

$\boxed{Pr[Preimage\ finding] \approx \frac{Q}{M}}$

Complexity of finding preimage $= \mathcal{O}(M)$