[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2023-2024
Scribed by: Archit Verma (202151192) — Lecture (Week 5)

AES (Advanced Encryption Standard)

AES is a block cipher used for symmetric encryption.AES is widely used today as it is a much stronger than DES and triple DES despite being harder to implement.

AES-128

- **Block Size**: AES-128 operates on blocks of data, each consisting of 128 bits.
- **Key Size**: The secret key used in AES-128 encryption and decryption is 128 bits long.
- **Number of Rounds**: The encryption process in AES-128 consists of 10 rounds, each round applying specific transformations to the data.
- **Secret Key**: AES-128 employs a 128-bit secret key, which is used to perform both encryption and decryption operations.

Rounds

Each round of AES encryption consists of the following operations:

1. **SubBytes**: This operation substitutes each byte in the state array with another byte using an S-box. It provides non-linearity and confusion in the encryption process by transforming the input data.
2. **ShiftRows**: In this operation, a cyclic left shift is performed on each row of the state array. The number of shifts is determined by the row index, ensuring diffusion of data within the block and improving resistance against cryptographic attacks.
3. **MixColumns** (except for the last round): The MixColumns operation transforms each column of the state array using matrix multiplication, further contributing to diffusion and confusion. It spreads the input data across the output, enhancing the security of the encryption process. However, this operation is omitted in the last round to simplify the decryption process without compromising security.

SubBytes

- The SubBytes transformation is a fundamental operation in AES, responsible for introducing non-linearity and confusion into the encryption process.
- It substitutes each byte in the state array with another byte using an S-box, which is a predefined lookup table.
- Transformation: $S : \{0,1\}^8 \rightarrow \{0,1\}^8$.

- The input byte x is represented as $x = X_0 X_1 \dots X_7$, where each X_i represents a binary digit (0 or 1) corresponding to the individual bits of the byte.
- The output byte y is computed using the S-box lookup table: $y = S(x)$.
- The S-box is constructed using a mathematical algorithm that involves finding the multiplicative inverse of polynomials modulo a fixed polynomial. This construction ensures good cryptographic properties and enhances the security of AES.
- The extended Euclidean algorithm is employed to find the multiplicative inverse, which is necessary for constructing the S-box and ensuring its cryptographic properties.

S-box

The S-box substitution is defined by the following transformation:

$$\begin{aligned} Y_0 &= S_1(X_0) \\ Y_1 &= S_1(X_1) \\ &\vdots \\ Y_7 &= S_1(X_7) \end{aligned}$$

where S_1 is a non-linear function.

S-box Construction

The S-box is constructed using a mathematical algorithm that involves finding the multiplicative inverses of polynomials modulo a fixed polynomial.

Extended Euclidean Algorithm

To find the multiplicative inverse of a polynomial $p(x)$ under modulo $x^8 + x^4 + x^3 + x + 1$, we use the extended Euclidean algorithm:

$$\begin{aligned} 1 &= \gcd(p(x), x^8 + x^4 + x^3 + x + 1) \\ &= p(x) \cdot q(x) + h(x^8 + x^4 + x^3 + x + 1) \end{aligned}$$

where $q(x)$ is the quotient polynomial and $h(x)$ is the remainder polynomial.

Illustrating SubByte Algorithm

To illustrate the SubBytes step, let's consider the input byte (01010011).

To represent the given input byte (01010011) as a polynomial, we assign each bit position a power of x in the polynomial representation. Starting from the left (most significant bit) to the right (least significant bit), we assign increasing powers of x to the bits where a 1 is present.

For the given input byte (01010011), the polynomial representation is derived as follows:

$$(01010011) = 0 \cdot x^7 + 1 \cdot x^6 + 0 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x^1 + 1 \cdot x^0 = x^6 + x^4 + x + 1$$

- We represent the input byte as a polynomial: $x^6 + x^4 + x + 1$. This representation is derived from the bit positions where 1s are present.

- We have an S-box transformation, $S : \{0,1\}^8 \to \{0,1\}^8$, which substitutes each byte using a lookup table.
- We perform the substitution: $S(01010011) = (11001010)$.

Now, let's compute $b_7 b_6 \dots b_0$:

$$\begin{aligned}
b_0 &= (a_0 + a_4 + a_5 + a_6 + a_7 + c_0) \mod 2 \\
&= (0 + 0 + 0 + 1 + 1 + 1) \mod 2 \\
&= 1 \\
b_1 &= 0 \\
&\vdots \\
b_7 &= 1
\end{aligned}$$

Thus, $b_7 b_6 b_5 \dots b_0 = 11101101$.
Finally, we find the output byte from the lookup table:

$$\text{subbyte}(53) = \text{ED}$$

To improve efficiency, we can store the calculations beforehand in a lookup table.

ShiftRows

- The ShiftRows operation performs a cyclic left shift on each row of the state array, ensuring diffusion of data within the block.
- It is essential for providing diffusion and improving the resistance against cryptographic attacks.
- Example:

$$\begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 & b_7 \\ b_8 & b_9 & b_{10} & b_{11} \\ b_{12} & b_{13} & b_{14} & b_{15} \end{bmatrix} \to \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_5 & b_6 & b_7 & b_4 \\ b_{10} & b_{11} & b_8 & b_9 \\ b_{15} & b_{12} & b_{13} & b_{14} \end{bmatrix}$$

MixColumns

- The MixColumns operation transforms each column of the state array using matrix multiplication, further contributing to diffusion and confusion.
- It involves multiplying each column with a fixed matrix to ensure the spread of input data across the output, enhancing the security of the encryption process.
- Example:

$$\begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 & b_7 \\ b_8 & b_9 & b_{10} & b_{11} \\ b_{12} & b_{13} & b_{14} & b_{15} \end{bmatrix} \to \begin{bmatrix} b'_0 & b'_1 & b'_2 & b'_3 \\ b'_4 & b'_5 & b'_6 & b'_7 \\ b'_8 & b'_9 & b'_{10} & b'_{11} \\ b'_{12} & b'_{13} & b'_{14} & b'_{15} \end{bmatrix}$$

Last Round

- The last round differs from the others as it does not include the MixColumns operation, simplifying the decryption process without compromising security. This omission reduces the complexity of decryption algorithms while maintaining the same level of security.

Key Schedule

- The Key Schedule generates round keys from the main secret key to be used in each round of the AES encryption process.
- It involves applying a series of transformations to the main secret key to derive subkeys for each round, ensuring the security of the encryption process.

Key Schedule Algorithm (KSA)

The Key Schedule Algorithm is responsible for generating round keys from the main secret key.

Left Circular Shift

The function `ROTWORD(B0,B1,B2,B3)` performs a left circular shift on the input bytes, resulting in $(B1, B2, B3, B0)$.

Size of B_i

Each B_i is 8 bits.

Subword Transformation

The `subword` transformation converts each byte B_i to B_i' using the SubBytes operation.

Constants for Round Keys

There are 10 constants used for the 11 round keys: $Rcon[1] = 01000000$, $Rcon[2] = 36000000$, and so on.

Key Expansion Algorithm

The key expansion algorithm computes the round keys from the main secret key. It iterates through 44 words (each word being 32 bits or 4 bytes) to generate the round keys.

The resulting round keys are $w[0] \ldots w[43]$, where each key is 128 bits.

Modes of Operation

- ECB (Electronic Codebook)
- CBC (Cipher Block Chaining)
- CFB (Cipher Feedback)
- OFB (Output Feedback)
- IGE (Infinite Garble Extension)

Algorithm 1 Key Expansion Algorithm

```
1: $w[0] \dots w[3]$ = Key
2: for $i = 4$ to 43 do
3:     temp $\leftarrow w[i-1]$
4:     if $i \mod 4 = 0$ then
5:         temp $\leftarrow$ subword(ROTWORD(temp)) $\oplus$ Rcon[i/4]
6:     end if
7:     $w[i] \leftarrow w[i-4] \oplus$ temp
8: end for
9: return $w[0] \dots w[43]$
```

ECB (Electronic Codebook)

ECB mode encrypts each block of plaintext separately using the same key.

Encryption in ECB

The encryption process in ECB mode involves dividing the plaintext into blocks, applying the encryption algorithm to each block using the secret key, and concatenating the resulting ciphertext blocks.

Decryption in ECB

The decryption process in ECB mode is the inverse of encryption, where each ciphertext block is decrypted using the same secret key.

Mathematical Representation

Let P be the plaintext, C be the ciphertext, and K be the secret key. Then, the encryption and decryption functions can be represented as:

$$C_i = \text{AES-128-Encrypt}(P_i, K)$$

$$P_i = \text{AES-128-Decrypt}(C_i, K)$$

CBC (Cipher Block Chaining)

CBC mode encrypts each plaintext block by XORing it with the previous ciphertext block before encryption.

Encryption in CBC

The encryption process in CBC mode involves XORing each plaintext block with the previous ciphertext block (or the initialization vector for the first block), encrypting the result using the secret key, and then updating the ciphertext block for the next iteration.

Decryption in CBC

The decryption process in CBC mode is similar to encryption, but it involves decrypting each ciphertext block and XORing it with the previous ciphertext block (or the initialization vector for the first block) to obtain the plaintext.

Mathematical Representation

Let IV be the initialization vector. Then, the encryption and decryption functions in CBC mode can be represented as:

$$C_i = \text{AES-128-Encrypt}(P_i \oplus C_{i-1}, K)$$
$$P_i = \text{AES-128-Decrypt}(C_i, K) \oplus C_{i-1}$$

Stream Ciphers

Stream ciphers are a type of symmetric encryption algorithm that encrypts plaintext data one bit or byte at a time, typically in a continuous stream. They are often used in situations where the plaintext data is transmitted continuously, such as in real-time communication systems like voice or video calls, or in applications where low latency is critical.

Overview

Let the message be a sequence of bits:

$$01010100\,01111000$$

Let

$$10101000\,01111000$$

be a sequence of random bits.

The ciphertext is obtained by performing bitwise XOR operation between the message and the random sequence:

$$01010100 \oplus 10101000 = 11111100$$
$$01111000 \oplus 01111000 = 00000000$$

So, the ciphertext is:

$$11111100\,00000000$$

where

$$10101000\,01111000$$

is the random sequence.

Perfect Secrecy: Given a ciphertext, the message can be any binary string of length 8.

Impracticalities:

1. Difficult to get random bits.
2. Sequence must be pre-distributed to both sender and receiver.
3. Requires a random sequence as long as the message.

Additive Stream Ciphers

Additive stream ciphers use a pseudorandom generator (PRG) to produce the pseudorandom sequence. The PRG extends a short "seed" into a long "pseudorandom" string, i.e.,

$$\text{PRG}(\mathit{seed}) = \text{pseudorandom sequence}$$

The seed is the secret key. The security of the additive stream cipher depends on the design of the PRG.

Security of PRG

PRG must be unpredictable for cryptographic use. This includes:

1. Next bit test: Given an initial segment, it should not be possible to efficiently guess the next bit.

2. Statistical Tests: The generated pseudorandom sequence should pass all polynomial time statistical tests.

The above notions are equivalent for stream cipher use.