|  |  |
| --- | --- |
| Ex.No.6 | **IMPLEMENTATION OF KEY GENERATION IN ADVANCED ENCRYPTION STANDARD** |

|  |
| --- |
| **AIM:** |

To implement key generation in Advanced Encryption Standard using Java/Python

|  |
| --- |
| **ALGORITHM:** |

**Initialize S-box and RCON:**

* Define the S-box (substitution box) and round constants (RCON) used for the key expansion.

**Convert Hex String to Byte Array:**

* **Input:** A hexadecimal string representing the AES key.
* **Output:** A byte array of the key.
* **Steps:**
  1. Initialize an empty byte array keyBytes of size half the length of the hexadecimal string.
  2. For each pair of hexadecimal characters, convert them to a byte and store them in keyBytes.

**Split Byte Array into Words:**

* **Input:** The byte array keyBytes.
* **Output:** An array of words where each word is 4 bytes.
* **Steps:**
  1. Create a 2D array words with 4 columns and the number of words needed (based on key size).
  2. For each word index, copy 4 bytes from keyBytes into the corresponding position in words.

**Generate New Words:**

* **Input:** Existing words and RCON.
* **Output:** New words generated using the AES key schedule.
* **Steps:**
  1. For each round, perform the following:
     + Rotate: Rotate the previous word (e.g., w3) to create a new word.
     + Substitute Bytes: Substitute bytes of the rotated word using the S-box.
     + XOR with RCON: XOR the substituted word with the round constant RCON.
     + Generate New Words: XOR the result with the word from the previous column to generate a new word.
     + Generate Additional Words: Use XOR operations to create additional words based on the previous words.
  2. Continue until all necessary words (e.g., w4 to w43) are generated.

**Print Results:**

* **Input:** The array of words generated during the key expansion.
* **Output:** The words in a matrix format.
* **Steps:**
  1. Print the words in a formatted matrix, where each column represents a round of key expansion.
  2. Display the hex representation of each word.

|  |
| --- |
| **CODING:** |

import java.util.Scanner;

public class AESKeyGeneration2 {

private static final String[][] SBOX = {

{"63", "7C", "77", "7B", "F2", "6B", "6F", "C5", "30", "01", "67", "2B", "FE", "D7", "AB", "76"},

{"CA", "82", "C9", "7D", "FA", "59", "47", "F0", "AD", "D4", "A2", "AF", "9C", "A4", "72", "C0"},

{"B7", "FD", "93", "26", "36", "3F", "F7", "CC", "34", "A5", "E5", "F1", "71", "D8", "31", "15"},

{"04", "C7", "23", "C3", "18", "96", "05", "9A", "07", "12", "80", "E2", "EB", "27", "B2", "75"},

{"09", "83", "2C", "1A", "1B", "6E", "5A", "A0", "52", "3B", "D6", "B3", "29", "E3", "2F", "84"},

{"53", "D1", "00", "ED", "20", "FC", "B1", "5B", "6A", "CB", "BE", "39", "4A", "4C", "58", "CF"},

{"D0", "EF", "AA", "FB", "43", "4D", "33", "85", "45", "F9", "02", "7F", "50", "3C", "9F", "A8"},

{"51", "A3", "40", "8F", "92", "9D", "38", "F5", "BC", "B6", "DA", "21", "10", "FF", "F3", "D2"},

{"CD", "0C", "13", "EC", "5F", "97", "44", "17", "C4", "A7", "7E", "3D", "64", "5D", "19", "73"},

{"60", "81", "4F", "DC", "22", "2A", "90", "88", "46", "EE", "B8", "14", "DE", "5E", "0B", "DB"},

{"E0", "32", "3A", "0A", "49", "06", "24", "5C", "C2", "D3", "AC", "62", "91", "95", "E4", "79"},

{"E7", "C8", "37", "6D", "8D", "D5", "4E", "A9", "6C", "56", "F4", "EA", "65", "7A", "AE", "08"},

{"BA", "78", "25", "2E", "1C", "A6", "B4", "C6", "E8", "DD", "74", "1F", "4B", "BD", "8B", "8A"},

{"70", "3E", "B5", "66", "48", "03", "F6", "0E", "61", "35", "57", "B9", "86", "C1", "1D", "9E"},

{"E1", "F8", "98", "11", "69", "D9", "8E", "94", "9B", "1E", "87", "E9", "CE", "55", "28", "DF"},

{"8C", "A1", "89", "0D", "BF", "E6", "42", "68", "41", "99", "2D", "0F", "B0", "54", "BB", "16"}

};

private static final String[] RCON = {

"01", "02", "04", "08", "10", "20", "40", "80", "1B", "36"

};

// Convert hex string to byte array

private static byte[] hexToBytes(String hex) {

int len = hex.length();

byte[] data = new byte[len / 2];

for (int i = 0; i < len; i += 2) {

data[i / 2] = (byte) ((Character.digit(hex.charAt(i), 16) << 4)

+ Character.digit(hex.charAt(i + 1), 16));

}

return data;

}

// Convert byte array to hex string

private static String bytesToHex(byte[] bytes) {

StringBuilder sb = new StringBuilder();

for (byte b : bytes) {

sb.append(String.format("%02X", b));

}

return sb.toString();

}

// Substitute bytes using the S-box

private static byte[] subBytes(byte[] word) {

byte[] substituted = new byte[4];

for (int i = 0; i < 4; i++) {

int row = (word[i] & 0xF0) >> 4;

int col = word[i] & 0x0F;

substituted[i] = (byte) Integer.parseInt(SBOX[row][col], 16);

}

return substituted;

}

// Rotate a word (used in key expansion)

private static byte[] rotate(byte[] word) {

byte[] rotated = new byte[4];

rotated[0] = word[1];

rotated[1] = word[2];

rotated[2] = word[3];

rotated[3] = word[0];

return rotated;

}

// XOR two byte arrays

private static byte[] xor(byte[] a, byte[] b) {

if (a.length != b.length) {

throw new IllegalArgumentException("Byte arrays must be of the same length.");

}

byte[] result = new byte[a.length];

for (int i = 0; i < a.length; i++) {

result[i] = (byte) (a[i] ^ b[i]);

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter input key (32 hexadecimal characters):");

String inputKey = scanner.nextLine();

// Ensure the input key is exactly 32 characters long (16 bytes for AES-128)

if (inputKey.length() != 32) {

System.out.println("The input key must be 32 hexadecimal characters long.");

return;

}

// Convert the input key to a byte array

byte[] keyBytes = hexToBytes(inputKey);

byte[][] words = new byte[44][4];

// Initialize the first 4 words (w0 to w3)

for (int i = 0; i < 4; i++) {

words[i] = new byte[] { keyBytes[4 \* i], keyBytes[4 \* i + 1], keyBytes[4 \* i + 2], keyBytes[4 \* i + 3] };

}

// Generate the remaining words (w4 to w43)

for (int i = 4; i < 44; i++) {

byte[] g = words[i - 1];

if (i % 4 == 0) {

g = subBytes(rotate(g));

byte[] rcon = new byte[] { (byte) Integer.parseInt(RCON[(i / 4) - 1], 16), 0, 0, 0 };

g = xor(g, rcon);

}

words[i] = xor(words[i - 4], g);

}

// Print the key expansion results

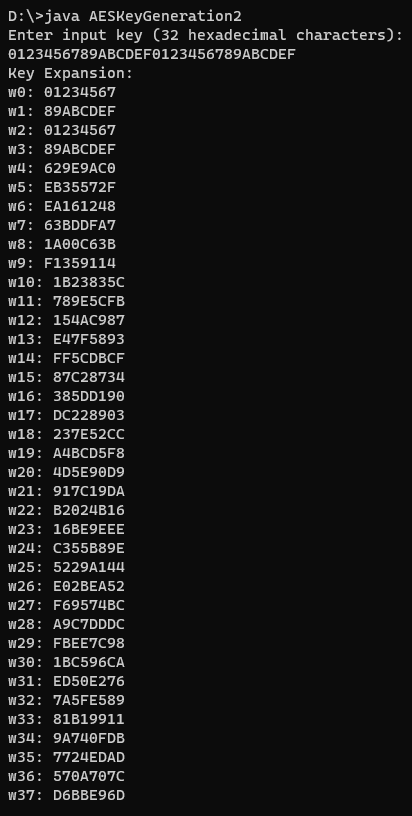
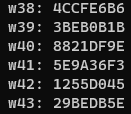
System.out.println("Key Expansion:");

for (int i = 0; i < words.length; i++) {

System.out.printf("w%d: %s\n", i, bytesToHex(words[i]));

} }}

|  |
| --- |
| **SCREEN SHOTS:** |

** **

|  |
| --- |
| **RESULT:** |

Thus I have implemented key generation in Advanced Encryption Standard using Java.

|  |
| --- |
| **Evaluation** |

|  |  |  |
| --- | --- | --- |
| Parameter | Max Marks | Marks Obtained |
| Uniqueness of the Code | 50 |  |
| Completion of experiment on time | 10 |  |
| Documentation | 15 |  |
| Total | 75 |  |
| Signature of the faculty with Date |  |  |