MIT-WPU
Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Faculty: Engineering and Technology

School of Computer Engineering & Technology
Programme: B.Tech Computer Sc. & Engineering

Name: Devanshu Surana
Roll No.: 23
Prn:1032210755
Panel: C batch:C1

Lab A3: Implement simplified AES symmetric key algorithm using python or java or C++

Objective of Lab

1. To understand symmetric encryption techniques, block cipher encryption and how the S-AES algorithm works

Theory

Symmetric encryption is a type of encryption where the same key is used for both the encryption and decryption processes. It's the most widely used form of encryption and is known for its efficiency in processing large volumes of data. One of the main components of symmetric encryption is the block cipher. Let's explore symmetric encryption techniques, block ciphers, and the Simplified Advanced Encryption Standard (S-AES) algorithm.

Symmetric Encryption Techniques:

Symmetric encryption techniques are used to secure data by using the same secret key for both encryption and decryption. Some key symmetric encryption techniques include:

1. Block Ciphers: Data is divided into fixed-size blocks, and each block is encrypted separately. Common block ciphers include AES, DES, and S-AES.

2. Stream Ciphers: Data is encrypted one bit or byte at a time, usually by combining it with a pseudorandom stream of bits generated from the key. RC4 is an example of a stream cipher.

3.Mode of Operation: Block ciphers are often used with various modes of operation (e.g., ECB, CBC, GCM) to provide additional security features such as confidentiality, integrity, and authenticity.

Block Cipher Encryption:

A block cipher is an encryption algorithm that operates on fixed-size blocks of data, typically 64 or 128 bits. It uses a secret key to transform the plaintext block into ciphertext and vice versa. The primary components of a block cipher are the key expansion, encryption, and decryption processes.

S-AES (Simplified Advanced Encryption Standard):

S-AES is a simplified version of the Advanced Encryption Standard (AES), designed for educational purposes and easy understanding of the AES algorithm. It uses a smaller block size and key size compared to AES. S-AES operates on 16-bit blocks and uses a 16-bit key.

Here's how S-AES works:

1. Key Expansion: S-AES generates round keys from the 16-bit key using a key expansion algorithm. These round keys are used in the subsequent rounds.

2. Initial Round: In the initial round, the input plaintext block is XORed with the round key.

3. Rounds (4 rounds): Each round consists of the following operations:
   -Substitution: S-AES employs a 4x4 S-Box for byte substitution.
   -Permutation: Rows and columns are shifted to achieve diffusion.
   -MixColumns: A linear transformation that mixes data across columns.

4. Final Round: Similar to the initial round but without the MixColumns operation.

5. Ciphertext: The final ciphertext is produced.

S-AES is a simplified version of AES, and its security is not as strong as the full AES, which is widely used for encryption in practice. Nonetheless, S-AES serves as a valuable learning tool for understanding the basic principles of block ciphers and encryption techniques. It provides insight into how key expansion, substitution, permutation, and mixing operations work in symmetric encryption algorithms.

Example of Using Simplified AES

Sure, let's go through an example of the Simplified Advanced Encryption Standard (S-AES) encryption and decryption process using a 16-bit block size and a 16-bit key. For simplicity, we'll use a fixed key and plaintext.

S-AES Key: 1101100110111010 (16 bits)

S-AES Plaintext: 1001011010110111 (16 bits)

S-AES Encryption:

Key Expansion: The key expansion process generates the round keys.

Initial key: 1101100110111010
Round 1 key: 0110111010110111
Round 2 key: 1101100110111010
Round 3 key: 0110111010110111
Round 4 key: 1101100110111010
Initial Round:

Plaintext: 1001011010110111
XOR with Round 1 key: 1001011010110111 ⊕ 0110111010110111 = 1111100000000000
Rounds (4 rounds):

Substitution, Permutation, MixColumns operations are applied in each round.
Final Round:

XOR with Round 4 key: 0000111010110110 ⊕ 1101100110111010 = 1101011100001100
Ciphertext: 1101011100001100

Algorithm:

S-AES Encryption:

Key Setup:

Key Expansion: S-AES starts with a 16-bit key and generates four 16-bit round keys. Each round key is derived from the original key through a key expansion process.
Initial Round:

Initial Round Key Addition: The plaintext block is XORed with the first round key.
Rounds (4 rounds in this example):

Substitution: Each 16-bit block is substituted using a 4x4 S-Box (Substitution Box).

Permutation: Rows and columns are shifted to achieve diffusion.

MixColumns: A linear transformation mixes data across columns.

Final Round:

Final Round Key Addition: The result from the last MixColumns operation is XORed with the fourth round key.

Code

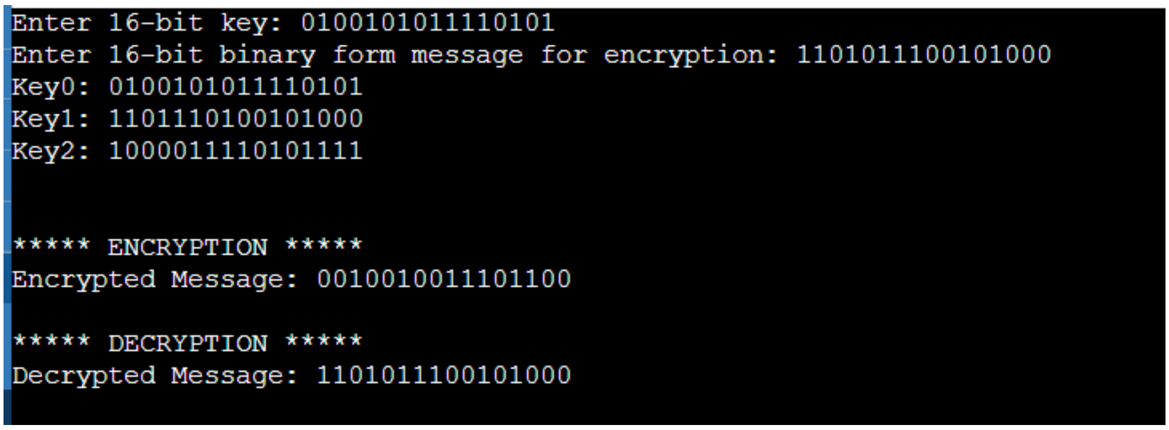
```
import java.util.Arrays;
import java.util.Scanner;
import java.util.*;
import java.util.regex.*;

public class SimplifiedAdvancedEncryptionStandard {

	private static final String[][] SBOX = {
{"1001","0100","1010","1011"},{"1101","0001","1000","0101"},{"0110","0010","0000","00
11"},{"1100","1110","1111","0111"} };
	private static final String[][] SBOX_INV = {
{"1010","0101","1001","1011"},{"0001","0111","1000","1111"},{"0110","0000","0010","00
11"},{"1100","0100","1101","1110"} };
	private static String key0 = null, key1 = null, key2 = null;
	private static int encryptionConstantMatrix[][] = { {1, 4}, {4, 1} };
	private static int decryptionConstantMatrix[][] = { {9, 2}, {2, 9} };

	public SimplifiedAdvancedEncryptionStandard(String key) {
		generateKeys(key);
	}

	private int binaryToDecimal(String binary) {
		return Integer.parseInt(binary, 2);
	}

	private String decimalToBinary(int decimal, int binaryStringSize) {
		return String.format("%" + binaryStringSize + "s",
Integer.toBinaryString(decimal & 0xFF)).replace(' ', '0');
	}

	public String stringXOR(String a, String b) {
		StringBuilder sb = new StringBuilder();
		for(int i = 0; i < a.length(); i++) {
		sb.append(a.charAt(i) ^ b.charAt(i));
		}
		return sb.toString();
	}
	private int gfMul(int a, int b) {
        int product = 0; //the product of the multiplication

        while (b > 0) {
          if ((b & 1) != 0) //if b is odd then add the first num i.e a into product result
            product = product ^ a;

          a = a << 1; //double first num

          //if a overflows beyond 4th bit
          if ((a & (1 << 4)) != 0)
            a = a ^ 0b10011; // XOR with irreducible polynomial with high term eliminated

          b = b >> 1; //reduce second num
        }
        return product;
      }

          private String nibbleSubstitution(String input, String[][] SBOX) {
                  StringBuilder sb = new StringBuilder();
                  for(int i = 0 ; i < input.length() / 4 ; i++) {
                          String str = input.substring(i*4, (i*4)+4);

          sb.append(SBOX[binaryToDecimal(str.substring(0,2))][binaryToDecimal(str.substrin
g(2,4))]);
                  }
                  return sb.toString();
          }

          private String shiftRow(String str) {
                  // Swap 2nd and 4th nibble
                  StringBuilder sb = new StringBuilder();
                  sb.append(str.substring(0,4));
                  sb.append(str.substring(12, 16));
                  sb.append(str.substring(8,12));
                  sb.append(str.substring(4,8));
                  return sb.toString();
          }

          private String rotateNibble(String word) {
                  return word.substring(4,8) + word.substring(0,4);
          }
```

```
	private void generateKeys(String key) {
		String w0 = key.substring(0,8);
		String w1 = key.substring(8,16);
		String w2 = stringXOR(stringXOR(w0, "10000000"),
nibbleSubstitution(rotateNibble(w1), SBOX));
		String w3 = stringXOR(w2, w1);
		String w4 = stringXOR(stringXOR(w2, "00110000"),
nibbleSubstitution(rotateNibble(w3), SBOX));
		String w5 = stringXOR(w4, w3);

		key0 = w0 + w1;
		key1 = w2 + w3;
		key2 = w4 + w5;
	}

	private String getKeys() {
		StringBuilder sb = new StringBuilder();
		sb.append("Key0: "+key0 + "\n");
		sb.append("Key1: "+key1 + "\n");
		sb.append("Key2: "+key2 + "\n");
		return sb.toString();
	}

	public String encrypt(String plainText) {
		// Round 0 - Add Key
		String roundZeroResult = stringXOR(plainText, key0);
		// Round 1 - Nibble Substitution -> Shift Row -> Mix Columns -> Add Key
		String shiftRowResult = shiftRow(nibbleSubstitution(roundZeroResult,
SBOX));
		String matrix[][] = new String[2][2];
		matrix[0][0] = shiftRowResult.substring(0,4);
		matrix[0][1] = shiftRowResult.substring(8,12);
		matrix[1][0] = shiftRowResult.substring(4,8);
		matrix[1][1] = shiftRowResult.substring(12,16);

		StringBuilder sb = new StringBuilder();
		for(int i = 0 ; i < encryptionConstantMatrix.length ; i++) {
			for(int j = 0 ; j < matrix.length ; j++) {
				String tempResults[] = new String[2];
				for(int k = 0 ; k < 2 ; k++) {
					tempResults[k] =
decimalToBinary(gfMul(encryptionConstantMatrix[i][k],binaryToDecimal(matrix[k][j])), 4);
```

```
                    }
                    sb.append(stringXOR(tempResults[0], tempResults[1]));
                }
            }
            String res = sb.toString();
            String mixColumnsResult = res.substring(0,4) + res.substring(8,12) +
res.substring(4,8) + res.substring(12, 16);
            String roundOneResult = stringXOR(mixColumnsResult, key1);
            // Round 2 - Nibble Substitution -> Shift Row -> Add Key
            String roundTwoResult =
stringXOR(shiftRow(nibbleSubstitution(roundOneResult, SBOX)), key2);
            return roundTwoResult;
        }

        public String decrypt(String cipherText) {
            // Round 0 - Add Key
            String roundZeroResult = stringXOR(cipherText, key2);
            // Round 1 - Shift Row -> Nibble Substitution -> Add Key -> Mix Columns
            String addKeyResult =
stringXOR(nibbleSubstitution(shiftRow(roundZeroResult), SBOX_INV), key1);
            String matrix[][] = new String[2][2];
            matrix[0][0] = addKeyResult.substring(0,4);
            matrix[0][1] = addKeyResult.substring(8,12);
            matrix[1][0] = addKeyResult.substring(4,8);
            matrix[1][1] = addKeyResult.substring(12,16);

            StringBuilder sb = new StringBuilder();
            for(int i = 0 ; i < decryptionConstantMatrix.length ; i++) {
                for(int j = 0 ; j < matrix.length ; j++) {
                    String tempResults[] = new String[2];
                    for(int k = 0 ; k < 2 ; k++) {
                        tempResults[k] =
decimalToBinary(gfMul(decryptionConstantMatrix[i][k],binaryToDecimal(matrix[k][j])), 4);
                    }
                    sb.append(stringXOR(tempResults[0], tempResults[1]));
                }
            }
            String res = sb.toString();
            String mixColumnsResult = res.substring(0,4) + res.substring(8,12) +
res.substring(4,8) + res.substring(12, 16);
            // Round 2 - Shift Row -> Nibble Substitution -> Add Key
```

```
		String roundTwoResult =
stringXOR(nibbleSubstitution(shiftRow(mixColumnsResult), SBOX_INV), key0);
		return roundTwoResult;
	}

	public static void main(String[] args) {
		String key = null, msg = null;
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter 16-bit key: ");
		key = sc.next();

		System.out.print("Enter 16-bit binary form message for encryption: ");
		msg = sc.next();

		SimplifiedAdvancedEncryptionStandard
simplifiedAdvancedEncryptionStandard = new
SimplifiedAdvancedEncryptionStandard(key);
		System.out.println(simplifiedAdvancedEncryptionStandard.getKeys());

		System.out.println("\n***** ENCRYPTION *****");
		String encryptedMsg = simplifiedAdvancedEncryptionStandard.encrypt(msg);
		System.out.println("Encrypted Message: "+encryptedMsg);

		System.out.println("\n***** DECRYPTION *****");
		String decryptedMsg =
simplifiedAdvancedEncryptionStandard.decrypt(encryptedMsg);
		System.out.println("Decrypted Message: "+decryptedMsg);
	}
}
```

Output Screen shots



```
Enter 16-bit key: 0100101011110101
Enter 16-bit binary form message for encryption: 1101011100101000
Key0: 0100101011110101
Key1: 1101110100101000
Key2: 1000011110101111


***** ENCRYPTION *****
Encrypted Message: 0010010011101100

***** DECRYPTION *****
Decrypted Message: 1101011100101000
```

Conclusion:

Thus, we have successfully learned and implemented Simplified AES cipher.

FAQs:

1. What is S-AES algorithm and how it is different from AES algorithm
2. Explain Key generation in S-AES
3. Explain Encryption in S-AES
4. Explain Decryption in S-AES