Practical 2

Aim: SDES Brute Force Attack Simulation

Description:
The provided code implements a brute-force attack on the Simplified Data Encryption Standard (SDES) in Java. SDES is a symmetric-key block cipher that operates on 8-bit plaintext blocks using a 10-bit key. The brute-force approach systematically tests all 1024 possible 10-bit keys to decrypt a given ciphertext and match it with a known plaintext.

The steps in the code are as follows:

1. **Key Generation:** All possible 10-bit keys are generated iteratively.
2. **Encryption and Decryption:** The SDES algorithm is used to encrypt plaintext and decrypt ciphertext using the specified key.
3. **Matching:** Each key is tested by decrypting the ciphertext and comparing it with the known plaintext.
4. **Output:** If the correct key is found, it is printed along with the decrypted text; otherwise, a failure message is displayed after testing all keys.

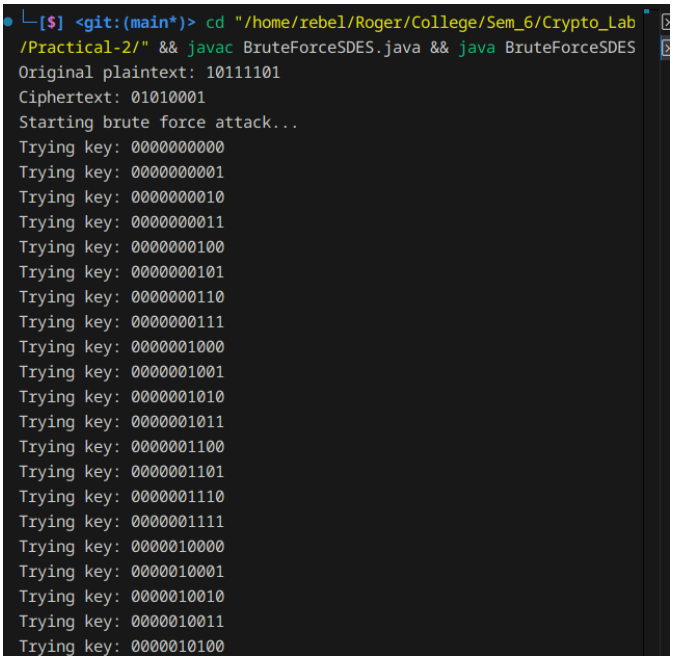
Code:

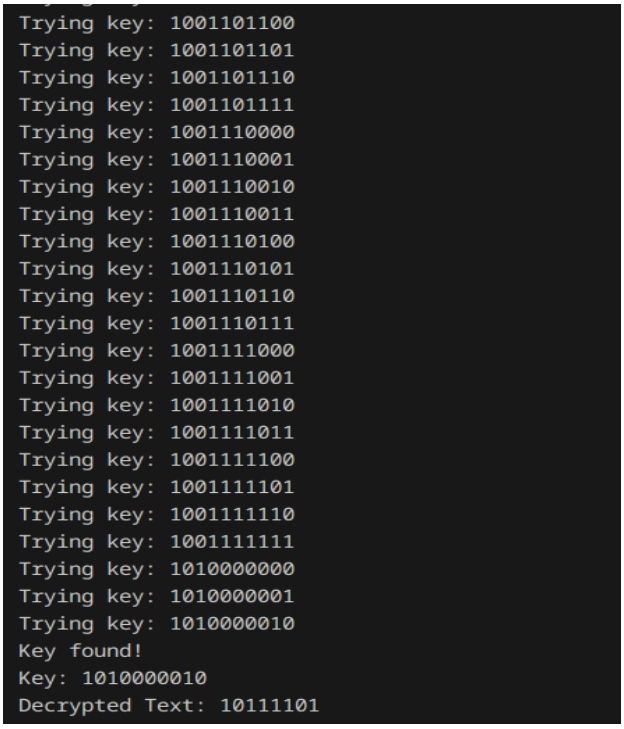
1. BruteForceSDES.java

```
import java.util.BitSet;

public class BruteForceSDES {

   // Function to brute force the key
   public static void bruteForce(BitSet ciphertext, BitSet knownPlaintext) {
      System.out.println("Starting brute force attack...");

      for (int i = 0; i < 1024; i++) {
         // Generate a 10-bit key from the loop index
         String binaryKey = String.format("%10s", Integer.toBinaryString(i)).replace(' ', '0');
         BitSet key = SDESEncryption.bitSetFromString(binaryKey, 10); // Use SDESEncryption

         BitSet decryptedText = SDESEncryption.decrypt(ciphertext, key); // Use SDESEncryption

         // Print each key
         System.out.print("Trying key: ");
         SDESEncryption.printBinary(key, 10); // Use SDESEncryption

         if (decryptedText.equals(knownPlaintext)) {
            System.out.println("Key found!");
            System.out.print("Key: ");
```

```
                SDESEncryption.printBinary(key, 10); // Use SDESEncryption
                System.out.print("Decrypted Text: ");
                SDESEncryption.printBinary(decryptedText, 8); // Use SDESEncryption
                return;
            }
        }
        System.out.println("Key not found within 1024 possibilities.");
    }

    public static void main(String[] args) {
        // Known key and plaintext for encryption
        BitSet key = SDESEncryption.bitSetFromString("1010000010", 10); // Use SDESEncryption
        BitSet plaintext = SDESEncryption.bitSetFromString("10111101", 8); // Use SDESEncryption

        System.out.print("Original plaintext: ");
        SDESEncryption.printBinary(plaintext, 8); // Use SDESEncryption

        BitSet ciphertext = SDESEncryption.encrypt(plaintext, key); // Use SDESEncryption
        System.out.print("Ciphertext: ");
        SDESEncryption.printBinary(ciphertext, 8); // Use SDESEncryption

        bruteForce(ciphertext, plaintext);
    }
}
```

Output:





Code Explanation:

1. **Key Generation:**
    - All 1024 possible 10-bit keys are systematically generated by iterating through the numbers 0 to 1023.
    - Each number is converted into a binary string and then into a `BitSet` representation for further use.
2. **Encryption and Ciphertext Generation**:
    - The known plaintext is encrypted using a predefined 10-bit key to generate the ciphertext, simulating an encrypted message.
3. **Brute-Force Attempt**:
    - Each 10-bit key is tested by attempting to decrypt the ciphertext.
    - Decryption involves using the SDES algorithm, which follows the standard two-round encryption process in reverse.
4. **Key Matching**:
    - The decrypted text is compared with the known plaintext for each key.
    - If a match is found, the corresponding key is identified as the encryption key.
5. **Output Results**:
    - If the correct key is found, the key and decrypted plaintext are displayed.
    - If no match is found after testing all 1024 keys, the program outputs a failure message.

Complexity Analysis:

Time Complexity:

1. **Key Generation**:
    - Generating each 10-bit key involves converting the loop index into a binary string and creating a `BitSet`.
    - This operation takes constant time for each key, **O(1)**.
    - As there are 1024 keys to test, the total time for key generation is **O(1024) = O(1)** (constant in practical terms, as the key space size is fixed).
2. **Decryption Process**:
    - For each key, the decryption process involves:
        - **Initial Permutation (IP)**: Constant time, **O(1)**.
        - **Two Rounds of fK**: Each round involves constant-time operations such as expansion, XOR, S-box substitutions, and permutation, making each round **O(1)**.
        - **Final Permutation (IP-1)**: Constant time, **O(1)**.
    - Since there are two rounds and additional permutations, the decryption process for one key is **O(1)**.
    - Testing all 1024 keys involves repeating the decryption process 1024 times, giving a total decryption time of **O(1024) = O(1)**.
3. **Comparison**:
    - Comparing the decrypted text with the known plaintext is a constant-time operation, **O(1)**, for each key.
    - Across 1024 keys, the total comparison time is **O(1024) = O(1)**.
4. **Overall Time Complexity**:
    - Combining the key generation, decryption, and comparison steps, the overall time complexity of the brute-force attack is **O(1024) = O(1)**, as the number of keys is fixed and does not grow with the input size.

Space Complexity:

1. **Storage for Keys**:
    - The current 10-bit key being tested is stored in constant space, **O(1)**.
2. **Input and Output Storage**:
    - The 8-bit ciphertext and known plaintext each require constant space, **O(1)**.
3. **Intermediate Values**:
    - Temporary values, such as permuted data, expanded data, and S-box outputs, require constant space, **O(1)**.
4. **Overall Space Complexity**:
    - The brute-force attack uses a constant amount of memory regardless of the number of keys, resulting in an overall space complexity of **O(1)**.