**Assessment rubric:**

Each of the two problems will be evaluated separately. The report for each project will contribute up to 20% of the total marks of the corresponding project. Note that the report should discuss the concepts, the vulnerabilities you found, the way you exploited it to design an attack, or optimize the attack, etc. and not discuss the specific implementation codes themselves. The code, along with accompanying comments should be self-explanatory, and this will account for up to 10% of the marks per project. Do feel free to cross-reference the comments with the report though. Even if your attack is not successful within 5 minutes, based on its consistency with the attack login described in the report and comments, you may earn up to 20% of the marks. The remaining 50% of the marks is essentially based on whether you manage to implement a successful attack in a time bounded manner, and its relative performance w.r.to other student’s submissions. Specifically, you can earn 20% of the marks if your attack is successful within 5 minutes (in our test environment: more details will be provided subsequently), while the remaining 30% of the marks will be based on the relative performance of solutions. Essentially, with a nice report and well documented code, which leads to a successful attack within 5 minutes, you will be able to secure up to 70% of the total marks for a project. The remaining 30% needs to be earned competitively, and is meant to both encourage you to go the extra mile w.r.to your classmates, and also to discourage students to share their solutions among each other. On that note, if we can identify cases of cheating or plagiarism, then the concerned students will not only be given zero marks for the project, but also further disciplinary actions will be taken.

**Problem Description:**

Consider a scenario where two principals, Alice and Bob, want to communicate with each other securely. Alice and Bob share a secret symmetric key 𝑘, which is known only to them and no one else. In order to avoid an attacker to perform cryptanalysis to uncover the secret key 𝑘, Alice and Bob try to minimize the availability of ciphertexts encrypted with 𝑘. They agree on a protocol to exchange a session key every time they want to communicate, using 𝑘 to encrypt relevant key materials to compute the session key. More specifically, the following are the assumptions and notations used in the following protocol:

- The protocol uses DES cipher in ECB mode.

- Alice and Bob (abbreviated as A and B below, respectively) share a secret key 𝑘, known only to them and no one else.

- Alice has a secret message 𝑚, which is known initially to her only, and which she wants to send confidentially to Bob using the protocol. We assume that m is a string of English characters and punctuation symbols. The string is at least 8 character long, and the first 8 characters are `[SECRET]’ (without the quotes), and we assume that the adversary is aware of this.

- We use the notation (𝑥, 𝑦) to denote the encryption of plain text 𝑦 with symmetric key 𝑥, using the DES cipher suite in ECB mode. Similarly, we use the notation (𝑥, 𝑦) to denote the decryption of ciphertext 𝑦 with key 𝑥.

- Given two bit sequences 𝑥 and 𝑦, we write (𝑥 ∥ 𝑦) to denote the bit sequence resulting from appending 𝑦 to 𝑥.

**Problem 1**

The protocol for exchanging secret messages consists of three steps, as explained below. STEP 1: Alice generates a 64-bit random number 𝑟, and sends it to Bob. The random number 𝑟 here serves as a challenge.

STEP 2: Bob receives a 64-bit message sent from Alice in step 1. Let us call this 𝑟’. (If the message actually comes from Alice, then 𝑟’ would be equal to 𝑟). Bob then generates a random session key 𝑠, also 64 bit long, and sends the following encrypted message back to Alice: 𝐸(𝑘, 𝑟′ ∥ ((𝑟′ + 1) ⨁𝑠)).

STEP 3: Alice receives a 128-bit message from Bob; let us call this 𝑋. Alice then does the following:

- Decrypts 𝑋 with the key 𝑘 to obtain: 𝑌 = (𝑘, 𝑋).

- Let 𝑈 be the first block of 64 bits of 𝑌, and 𝑉 be the second block of 64 bits of 𝑌.

- If 𝑈 ≠ 𝑟 then Alice knows the message has been altered so she aborts the protocol.

- Otherwise, Alice computes the session key 𝑠’ = (𝑟 + 1)⨁𝑉, and sends 𝐸(𝑠′ , 𝑚) to Bob.

Note that if the messages in Step 1 & 2 have not been altered then 𝑉 = (𝑟 + 1)⨁𝑠, and 𝑠′ = (𝑟 + 1)⨁𝑉 = (𝑟 + 1)⨁(𝑟 + 1)⨁𝑠 = 𝑠. So Alice would obtain the correct session key. At the conclusion of a normal run of the protocol, Bob would have received the secret 𝑚 encrypted with 𝑠, which he can then decrypt to obtain the plain text 𝑚. The protocol is illustrated in the following diagram, where A ⟶ B : M means Alice (A) sends message M to Bob (B).

1. A ⟶ B: 𝑟

2. B ⟶ A: (𝑘, 𝑟 ∥ ((𝑟 + 1)⨁𝑠))

3. A ⟶ B: (𝑠, 𝑚)

Note that the diagram only illustrates what should happen in a normal exchange between legitimate parties, i.e., Alice and Bob. You will assume the role of an intruder, who initially knows neither the secret key 𝑘 nor the secret message m, but for the fact that the message starts with [SECRET]. Your task is to find a way to break the protocol to recover the whole secret message 𝑚. You will do so by writing a Java program to automate the attack. It is not necessary to implement an actual network protocol. The behaviours of Alice and Bob will be simulated by Java programs as specified below. A project template containing these programs will be provided separately. To implement the attack, you will need to query these programs to either supply A and B with altered messages or to trick them to perform certain operations related to the protocol to achieve your goal.

**Project template:**

A project template (file Problem1.zip) will be provided along with this assignment. The template contains two classes: Alice and Bob, implementing, respectively, the roles of Alice and Bob in the protocol. A brief description of these two classes is given below; for more detailed descriptions, see the source codes of the classes in the project template. Both classes contain a private member sharedKey, which is the secret key 𝑘 shared by Alice and Bob, and a private member sessionKey, which represents the session key used in each session. Class Alice contains additionally another private member secretMessage, which represents the secret message m, and a private member random64bit, which represents the random number generated by Alice at each session of the protocol. The class Alice contains among others the following public methods, which correspond to Step 1 and Step 3 of the protocol.

• byte[] Step1(): This method generates a 64-bit random number and stores it in the private member 𝑟. It returns 𝑟 in a byte array of size 8.

• String Step3(byte[] input): This implements step 3 of the protocol. The input parameter must be an array of size 16. This method will first decrypt the input with the stored key 𝑘 in the class, and compares the first 64 bits of the decrypted message with the stored random value r created in Step1(). If they do not match, the function returns a null pointer. Otherwise it returns a string, which contains the secret message 𝑚 encrypted with the session key computed by Alice according to step 3 in the protocol. The class Bob contains the following public method:

• byte[] Step2(byte[] input): This implements step 2 of the protocol. The `input’ parameter must be a byte array of size 8, representing a 64-bit number. It returns a byte array of size 16, representing the message sent by Bob to Alice in Step 2 of the protocol. Your attack algorithm can only invoke public methods available in the classes in the template. In particular, your algorithm should not assume any knowledge of the secret key 𝑘, the session key s, and the secret message m (other than the fact that m starts with ‘[SECRET]’). When your attack is successful, you must print the message m to the standard output. A sample key 𝑘 and a sample message m are provided in the project template, but do note that these will be changed during the testing of your submission to prevent you from hardcoding the key and the message in your solution. In addition to the classes Alice and Bob above, a class containing helper functions, called CommonFunctions, is also included. These functions are used in the implementation of the protocol and you may use these to help implementing your attacks. The methods of the class are as follows:

• concat: Simply takes two byte arrays as input and concatenates them together. The resulting array size is the sum of the input array sizes.

• incrementByValue: takes a byte array and an integer value as input. Adds the integer value to the value of the byte array and returns the result in the byte array format.

• XOR: takes two byte arrays as input. The precondition is that the two byte arrays of the input are of the same size. If the precondition is satisfied, then the two byte arrays are XORed bit by bit and the function returns the resulting byte array.

• encrypt and decrypt: take a string and a cipher as input, respectively. The function encrypts/decrypts the string using the information (type of the encryption scheme, the key used, etc.) provided by the Cipher object.

• swap: simply takes a byte array and swaps the first and the last bytes of the byte array and returns the resulting byte array.

**Deliverable:**

Implement your attack in the class template Attack1 in the provided project template Problem1. Put all your codes in the class Attack1. You may change the values of the parameters of the protocols to help with your testing and debugging, but do not modify any other parts of the project template.

**Problem 2**

This problem is a small variation of Problem 1. The assumptions and the notations for the protocol are as in Problem 1. The only differences are in Step 1 of the protocol, where the random bit sequence is encrypted with 𝑘 before being sent, and Step 2, where the received message is decrypted first.

STEP 1: Alice generates a 64-bit random number 𝑟, and sends (𝑘, 𝑟) to Bob.

STEP 2: Bob receives a 64-bit message 𝑋 sent from Alice in step 1, and decrypts it with 𝑘 to obtain 𝑟’ = (𝑘, 𝑋). (If the message actually comes from Alice, then 𝑟’ would be equal to 𝑟). Bob then generates a random session key 𝑠, and sends the following encrypted message back to Alice: (𝑘, 𝑟′ ∥ ((𝑟′ + 1) ⨁𝑠)).

STEP 3: as in Problem 1. The following diagram illustrates what should happen in a normal exchange between Alice and Bob:

1. A ⟶ B: (𝑘, 𝑟)

2. B ⟶ A: (𝑘, 𝑟 ∥ ((𝑟 + 1)⨁𝑠))

3. A ⟶ B: (𝑠, 𝑚)

As in Problem 1, you are asked to implement an attack to break the protocol to obtain 𝑚, without initially knowing the secret key 𝑘, the session key 𝑠, or the secret message 𝑚.

**Project template:**

A project template (file Problem2.zip) will be provided along with this assignment. The template contains two classes: Alice and Bob, implementing, respectively, the roles of Alice and Bob in the protocol. The class members and methods are the same as in Problem 1, the only difference is in the implementation of Step 1 and Step 2.

**Deliverable:**

Implement your attack in the class template Attack2 in the provided project template Problem2. Put all your codes in the class Attack2. You may change the values of the parameters of the protocols to help with your testing and debugging, but do not modify any other parts of the project template