Birzeit University
Faculty of Engineering and Technology
Department of Electrical and Computer Engineering
ENCS5121 – Information Security and Computer Network Laboratory (Term 1242)
Project (*Phase-3*), Due Thursday, Jun. 12, 2025

A. Document Objectives

This document describes the lab project phase III, its deliverables, and its grading criteria.

B. Project - Phase III:

Description:

Modify the online guessing game application that you had for **Phase II** so that your code authenticates the client and the server to each other, then establishes a session (i.e., game round) key to be used by them to encrypt/decrypt the exchanged data between them. To achieve these objectives, implement the protocol provided here in this phase. When implemented properly, the provided protocol achieves mutual authentication, perfect forward secrecy (PFS), and is immune against man-in-the-middle (MITM) attacks.
For public-key cryptosystem operations:

1. Each side needs to use RSA with a minimum size of 310 decimal digits for each of the 2 required prime numbers, p and q. You can pick the 2 unique prime numbers for each side from (https://primes.utm.edu/curios/index.php?start=301&stop=1000) for a total of 4 prime numbers.
2. Subsequently, select a proper e for each side so that it is relatively prime to (p – 1)(q – 1), and find the multiplicative inverse, d.
3. To find a proper e, start by selecting a prime number (other than what was selected for p and q) and make sure that the greatest common devisor (GCD) between e and (p – 1)(q – 1) is 1 (you can use an online GCD tool for that or write your own separate code). If the GCD is not 1, then select another prime number for e and repeat the process until the GCD = 1.
4. To find d either write a separate code (https://stackoverflow.com/questions/4798654/modular-multiplicative-inverse-function-in-python). Make sure to use (p – 1)(q – 1) as the modulus when finding both e and d. Note that if you use the online calculator, then you need to compute (p – 1)(q – 1) first before using the result as the modulus.

5. Once e and d are found for each side, then, for simplicity, hard code the resultant public keys (N, e) for each side at both sides. As for the corresponding private key, you should store the encrypted version of the private key, d, at the owner's side. However, for simplicity, hard code the private key as a plaintext in the game guessing application of the owner's side. Note that, in real life, hard coding the private key is a totally insecure practice and should not be performed ☺.

Moreover, you need to select a generator, **g**, and a prime, **m**, for the Diffie-Hellman key exchange used by the given protocol. To select the proper values for **g** and **m**, use the values found in the "**2048-bit MODP Group**" of RFC3526 (https://www.ietf.org/rfc/rfc3526.txt). As explained in the RFC, higher MODP groups should be used for establishing a 256-bit key for AES. However, for simplicity, use the values found in the "2048-bit MODP Group". Both **g** and **m** should be hard coded in the game guessing application of both sides.

At the beginning of each game round, Alice (i.e., client) selects an exponent **a** and Bob (i.e., server) selects an exponent **b** using a cryptographically secure **RNG** function/method. Each exponent should be 2048-bits long. These exponents will be used by the Diffie-Hellman key exchange used by the provided protocol. Moreover, at the beginning of each game round, Alice selects a challenge, $\mathbf{R_A}$, and Bob selects a challenge, $\mathbf{R_B}$, using a cryptographically secure **RNG** function/method. Each of $\mathbf{R_A}$ and $\mathbf{R_B}$ should be 256-bits long. Note that the words "Alice" and "Bob" that are used in computing H in the provided protocol and in step 3 refer to unique IDs for Alice and Bob such as their IP addresses.

After step 2 of the protocol, Alice verifies the signature of $\mathbf{S_B}$ using Bob's public key, computes **H**, and compares it with **H** that was included in $\mathbf{S_B}$. If the result of the verification and the comparison is valid, Alice proceeds to step 3 of the protocol. Otherwise, Alice displays a message indicating that Bob is not authenticated before terminating the game round. Bob follows a similar approach after step 3 to authenticate Alice. Use SHA256 to compute **H**, and to compute the session key, **K**. In performing the public-key operations (i.e., signing and verifying the signature), repeated squaring must be used for efficiency. You can use the appropriate programming language code available at (https://rosettacode.org/wiki/Modular_exponentiation). At the end of step 2, both Alice and Bob must destroy the exponents **a** and **b** they selected at the beginning of the game round. At the end of step 3, both Alice and Bob have a shared session key, **K**, that replaces the hard coded key that you had in **Phase II** code. Hence, both Alice and Bob use **K** to encrypt/decrypt the exchanged data between them using the code you had for **Phase II** which uses AES-256-CBC. At the end of each game round, both Alice and Bob must destroy the established key K to preserve PFS.

Phase III must be your own genuine code. You may use existing libraries/methods/functions for SHA256 and for repeated squaring, but the rest of the code needed for **Phase III** must be developed by you.

Make sure to test your modified application preferably using 2 virtual machines. Verify your code by considering three test cases:

1. **Test case 1** – Normal test case: Execute 2 different game rounds.
2. **Test case 2** – Trudy posing as Bob: Intentionally change the private key **d** of Bob in the code to a different value. The execution of your code should result in Alice failing to authenticate Bob and displaying a message indicating that Bob is not authenticated before terminating the game round.
3. **Test case 3** – Trudy posing as Alice: Intentionally change the private key **d** of Alice in the code to a different value. The execution of your code should result in Bob failing to authenticate Alice and displaying a message indicating that Alice is not authenticated before terminating the game round.

C. Deliverables

Submit a well-documented soft copy of your **implementation** (**.py**), **video** file, a **readme file** (**.txt**) on how to execute your implementation through RITAJ in one .zip file (ID.zip → ***1191615.zip***) before the deadline. Make sure to state what code was added specifically for **phase III**. The documentation should provide the following:

1. Well-commented source code of your implementation, with clear highlights on the sections specifically added in this phase (**.py**).
2. A video containing the execution part of your application's testing step by step no more than **ten minutes**. You should clearly discuss the steps for each test case **by your voice**. The video should also clearly contain the following:
   a. **Test case 1**: Before destroying the exponents **a** and **b** that are selected at the beginning of each game round and the established key **K**, printout messages by each application showing the exponents **a** and **b**, the key **K**, the numbers **R_A** and **R_B**, confirmation that Bob is authenticated to Alice, and confirmation that Alice is authenticated to Bob. Provide also snapshots of the **IV** used for each exchanged data. Remember that you need to repeat this **for 2 different game rounds.**
   b. **Test case 2**: Printout messages by each application showing the exponents **a** and **b** selected at the beginning of the game round, confirmation that Bob is not authenticated to Alice, and the game round is being terminated.
   c. **Test case 3**: Printout messages by each application showing the exponents **a** and **b** selected at the beginning of the game round, confirmation that Alice is not authenticated to Bob, and the game round is being terminated.
3. A detailed **README** (.txt) file outlining the requirements for running your implementation, along with step-by-step instructions on how to execute and test it.

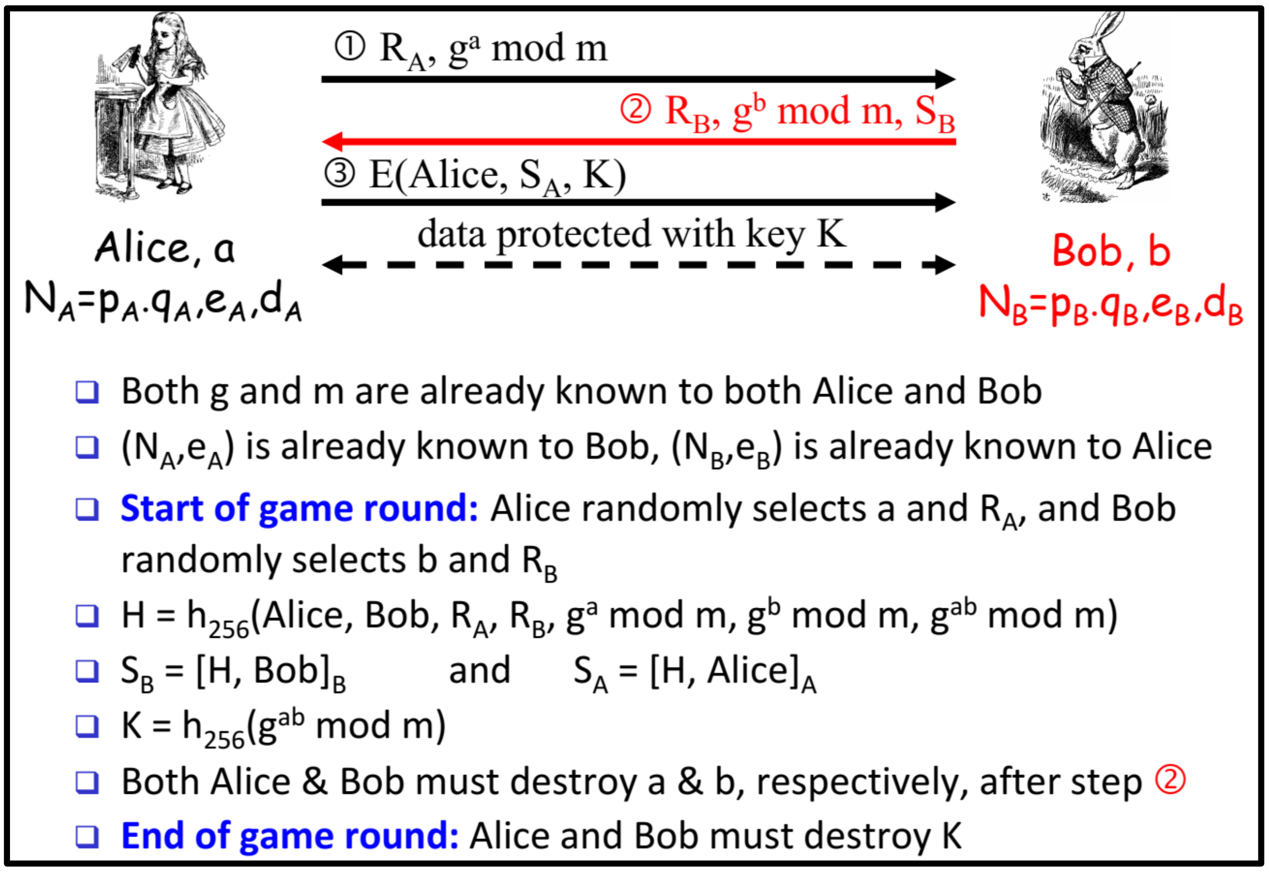
D. Grading Criteria

The following table summarizes the grading criteria of the lab project phase III:

Program readability and comments **(.py)**	10 pts.
Step-by-step readme file **(.txt)**	10 pts.
Program executes correctly over a network	20 pts.
Video:	
Test case 1	20 pts.
Test case 2	20 pts.
Test case 3	20 pts.
Total	**100 pts.**

Generally, just by following the guidelines presented in this document, you should get a good score in phase III. However, failing to stick to these guidelines may result in a reduction proportional in magnitude to the deviation.

Good luck, *ENCS5121 Team*


① R_A, g^a mod m
② R_B, g^b mod m, S_B
③ E(Alice, S_A, K)
data protected with key K
Alice, a
$N_A = p_A \cdot q_A, e_A, d_A$
Bob, b
$N_B = p_B \cdot q_B, e_B, d_B$

- Both g and m are already known to both Alice and Bob
- (N_A, e_A) is already known to Bob, (N_B, e_B) is already known to Alice
- **Start of game round:** Alice randomly selects a and R_A, and Bob randomly selects b and R_B
- $H = h_{256}(\text{Alice}, \text{Bob}, R_A, R_B, g^a \bmod m, g^b \bmod m, g^{ab} \bmod m)$
- $S_B = [H, \text{Bob}]_B$ and $S_A = [H, \text{Alice}]_A$
- $K = h_{256}(g^{ab} \bmod m)$
- Both Alice & Bob must destroy a & b, respectively, after step ②
- **End of game round:** Alice and Bob must destroy K