King Fahd University of Petroleum and Minerals
College of Computer Sciences and Engineering
Department of Computer Engineering

COE 451 – Computer and Network Security (T191)

Mini-Programming Project-Phase III (due date: Tuesday 10/12/2019)

Description:

Modify the file transfer application that you had for phase 2 so that your code authenticates the client and the server to each other, then establishes a session key to be used by them to encrypt/decrypt the files exchanged between them. To achieve these objectives, implement the protocol provided on the last page. When implemented properly, the provided protocol achieves mutual authentication, perfect forward secrecy (PFS), and is immune against man-in-the-middle (MiM) attacks.

For public-key cryptosystem operations, 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. 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*. To find *d* either write a separate code (Python: https://stackoverflow.com/questions/4798654/modular-multiplicative-inverse-function-in-python, Java: https://www.tutorialspoint.com/java/math/biginteger_modinverse.htm), or use the online calculator (https://www.dcode.fr/modular-inverse). **Make sure to use $(p - 1)(q - 1)$ as the modulus**. 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. 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 file transfer 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 file transfer application of both sides.

At the beginning of each session of the file transfer application, 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 session of the file transfer application, Alice selects a challenge, R_A, and Bob selects a challenge, R_B, using a cryptographically secure RNG function/method. Each of R_A and 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 S_B using Bob's public key, computes H, and compares it with H that was included in 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 session. 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 session. 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 the mini-project phase (2) code. Hence, both Alice and Bob use K to encrypt/decrypt the files exchanged between them using the code you had for mini-project phase (2) which uses AES-256-CBC. At the end of each session, both Alice and Bob must destroy the established key K to preserve PFS.

Phase 3 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 3 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:

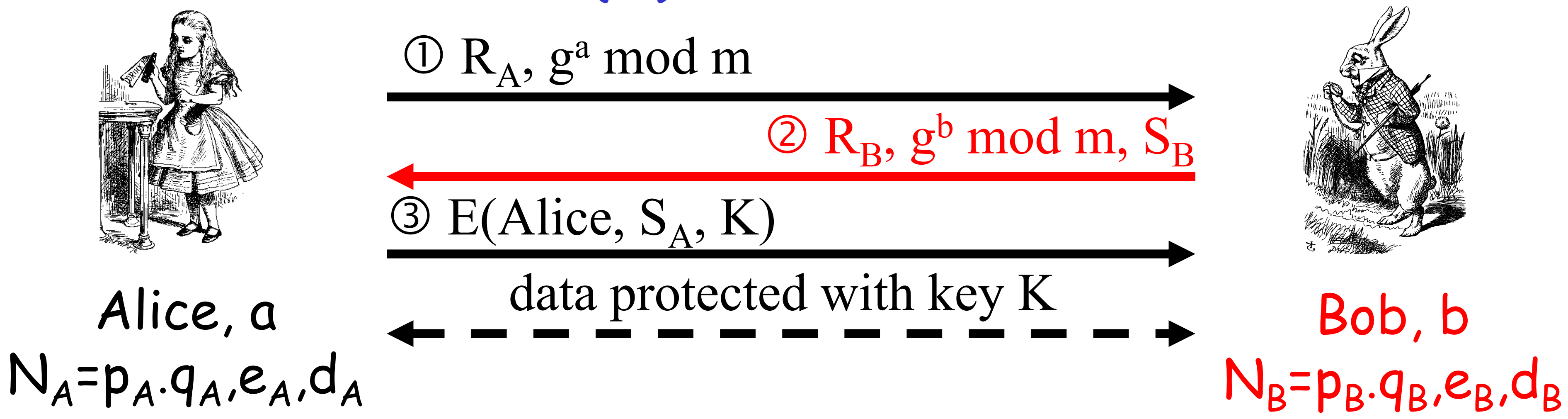
1. **Test case 1** – Normal test case: Execute 3 different file transfer sessions.
2. **Test case 2** – Trudy posing as Bob: Intentionally change the private key d of Bob in the code to a different value. The execution of your code should result in Alice failing to authenticate Bob and displaying a message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

Deliverables:

Submit a **well-documented** soft copy of your implementation along with a **readme file** on how to execute your implementation on Blackboard. Make sure to state what code was added specifically for phase 3. The **documentation** should provide the following:

1. **Test case 1:** Before destroying the exponents a and b that are selected at the beginning of each session and the established key K, printout messages by each file transfer 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 file.
2. **Test case 2:** Printout messages by each file transfer application showing the exponents a and b selected at the beginning of the session, confirmation that Bob is not authenticated to Alice, and the session is being terminated.
3. **Test case 3:** Printout messages by each file transfer application showing the exponents a and b selected at the beginning of the session, confirmation that Alice is not authenticated to Bob, and the session is being terminated.

Phase (3): Modified SSH



- 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 session:** Alice randomly selects a and R_A, and Bob randomly selects b and R_B
- H = h_{256}(Alice, Bob, R_A, R_B, g^a mod m, g^b mod m, g^{ab} mod m)
- S_B = [H, Bob]$_B$ and S_A = [H, Alice]$_A$
- K = h_{256}(g^{ab} mod m)
- Both Alice & Bob must destroy a & b, respectively, after step ②
- **End of session:** Alice and Bob must destroy K