ECE 443/518 Fall 2022 - Project 3

Perfect Forward Secrecy with Double RSA

Report Due: 11/13 (Sun.), by the end of the day (Chicago time)
Late submissions will NOT be graded

I. Overview

While perfect forward secrecy (PFS) can be achieved efficiently by using DHKE and its variants, it is possible and convenient to use the RSA algorithm only to implement a PFS system. As mentioned in our lectures, the RSA algorithm need to be applied twice, and below is a possible protocol.

1. Bob generates a RSA key pair (AuthPublic, AuthPrivate) for authentication only and deliver it to Alice through an authentic channel.
2. When Alice needs to send a message to Bob, Bob generates a RSA key pair (EncPublic, EncPrivate) as the session key.
3. Bob calulates the signature EncPubSig of EncPublic with the authentication key AuthPrivate, and delivers (EncPublic, EncPubSig) to Alice.
4. Alice verifies (EncPublic, EncPubSig) with AuthPublic to make sure she is communicating with Bob.
5. Alice encrypts the plaintext using EncPublic and sends the ciphertext to Bob.
6. Bob decrypts the ciphertext to obtain the plaintext.

For this project, we are going to use the rsa package in Go to implement the above protocol. In particular, Probabilistic Signature Scheme (PSS) is used for RSA signature and Optimal Asymmetric Encryption Padding (OAEP) is used for RSA encryption. For your conveniece, examples and part of the protocol are provided here. Now it is your turn to complete the protocol and to test it using knowledge of public-key cryptography and digital signature.

II. RSA Signature and Encryption

Before you actually write code for the protocol, you will first need to understand how to sign/verify messages in RSA using PSS and how to encrypt/decrypt messages in RSA using OAEP in Go. You will need to read the two functions validateRSAPSS() and validateRSAOAEP() and to observe their outputs in order to understand how they work.

III. Implement the Protocol

The implementation of our double RSA protocol starts with the function doubleRSAPFS(), which calls a few more functions, each implements one or more protocol steps. You will need to modify the three functions aliceVerify, aliceEncrypt, and bobDecrypt for the last 3 steps since the implementation of the first 3 steps are already provided. Once the protocol is implemented correctly, doubleRSAPFS() will output "doubleRSAPFS completed successfully."

Moreover, since our protocol may also prevent certain cases of man-in-the-middle (MITM) attacks, the function doubleRSAPFSMITM() simulates a possible attack scenario. Your implementation of aliceVerify, aliceEncrypt, and bobDecrypt, without modification, should be able to protect against this attack so that the function should output "doubleRSAPFSMITM completed successfully."

Now, you are ready to reason further with our double RSA protocol and answer the following 4 questions in your project report:

1. Why do we need to apply RSA twice? Why can't Bob simply generate a RSA encryption key everytime Alice wants to send a message and skip the authentication/signature part?
2. For step 3, when Bob delivers (EncPublic, EncPubSig) to Alice, what kind of channel is needed? A secure channel? An authentic channel? Or an insecure channel is perfectly OK?
3. Can Alice use the protocol to send Bob a message of arbitrary length? If not, how can you modify the protocol to achieve so?
4. Is it possible for Oscar to perform a man-in-the-middle attack on the protocol to pretend that he or she is Alice? Why or why not? If not, how can you modify the protocol to prevent so?

IV. Deliverables

Submit the following to Blackboard for this project.

1. 50 Points: Your source code that can be executed correctly.
2. A project report including the items below.
    - 10 Points: explain your approach and results for Section III.
    - 40 Points: answers to the four questions in Section III.

The project should be done individually. You can discuss the project with other students but all the source code and writings should be your OWN. PLAGIARISM and called for DISCIPLINARY ACTION. NEVER share your source code and reports with others.