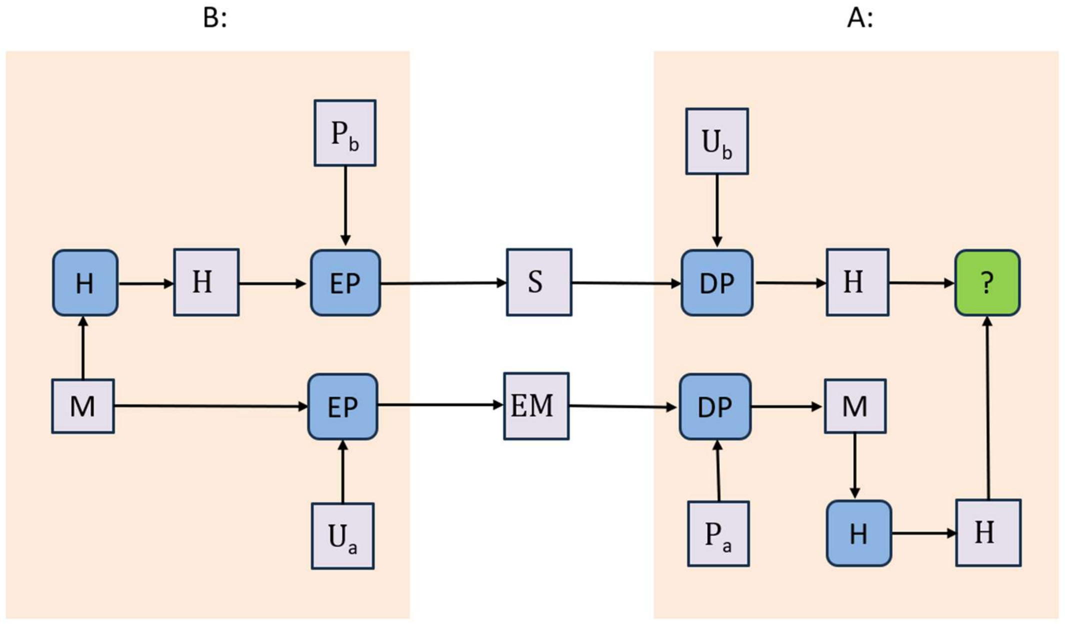
COSC 3371 Cybersecurity
Homework Assignment 3
COSC 3371, Spring 2024

Version 1, March 30, 2024
Version 2, April 3, 2024

This assignment focuses on several essential concepts discussed recently. These topics include encryption (public-key encryption, RSA encryption) for privacy, hashing that produces digests for integrity, and authentication of the sender's identity. The overall structure of the assignment, discussed in class recently, is shown below (parallel to our notations used in the class slides). RSA provides some functions to simplify the work, such as generating and verifying the signature. Each implementation of the cryptography module may be slightly different. For example, RSA combines hashing (H) and encryption (EP) into one function and provides one function to verify the signature.



P: Private Key, U: pUblic Key

User B would like to send a message to User A securely. The requirements are:
- The message must be encrypted using public-key encryption (RSA).
- The message must also produce a hash for integrity check.
- The hash must be signed to ensure it comes from B.
- There is no need to concatenate the signature and the encrypted message.

There are several places where you have choices. Here are the preferred parameters.
- Key size of the public and private key: 2048
- Hash: SHA-256
- Signing the hash (sign_hash).

The assignment must use the **RSA** module in **Python**. You must install it in your Python interpreter if it has not already been installed. Many references are available online, such as https://stuvel.eu/python-rsa-doc/. The module provides functions to generate public-private key pairs, hash, encryption, and verification. If you encounter difficulty using RSA, please get in touch with the TA for help.

We are NOT going to test the program across the network. Instead, we will simulate it in a Python program. To do so, we will create two classes, sender and receiver, to separate the two sides.

Sender class:

- A Sender will hold the following data: the plaintext message, the pair of keys, the hash value (digest), the signature, and the ciphertext. Set them to None initially.
- Sender's methods:
    - The constructor __init__,
    - The encrypt function
    - The sign function

Receiver class:

- A Receiver will hold the following data: the plaintext message, the pair of keys, the signature, and the ciphertext. Set them to None initially.
- Receiver's methods:
    - The constructor __init__,
    - The decrypt function
    - The verify function
-

Problem 1 (30 Points): Successful Validation.

You will input a message and hash, sign, and encrypt it according to the description above. Ensure each user (A and B) can access its private key. Public keys are available for both. We assume there is no malicious activity related to this part. Whenever possible, print an explanation of what you did and the output of any value produced.

Submission: One Python file.

Problem 2 (10 Points): Simulated Attack.

The two classes defined in Problem 1 should remain the same for this part. You are modifying the main program to experiment. The experiment simulates an attack with someone changing the message after signing it. We would like to see if the code can detect this attack. In reality, it is hard to edit an encrypted message. Here is the suggestion: Take the plaintext file, swap two characters, and encrypt the modified file. Your signature should be generated with the text in the original plaintext file.

Submission: One Python file, modified from Problem 1. The class definitions should be the same.

Problem 3 (10 Points): Simulated Attack.

For this part, simulate an attack on the signature (as opposed to the message like in Problem 2). You may propose more than one way to simulate the attack.

Submission:

- One Python file, modified from Problem 1. The class definitions should be the same.
- A one-page PDF file explaining what you did and your conclusion (for Problems 1-3).