COSC 3371 Cybersecurity - Assignment 4

Digital Signature

Course	Points	Language	Due Date
COSC 3371 Cybersecurity	100 pts	Python 3.10+ | cryptography library required	April 27, 2026

1. Overview

In this assignment, you will use the Hash-then-Sign approach to show how digital signatures provide integrity and non-repudiation with public-key cryptography. Both parts are built as a client–server application. Each part provides four skeleton files (one sender and one receiver) that you must complete by filling in the TODO placeholders, running the programs, and explaining the observed behavior.

2. Learning Objectives

- Describe the Hash-then-Sign process used in digital signatures.
- Implement sender and receiver logic from partially completed skeleton code.
- Demonstrate one accepted case and one rejected case.
- Interpret verification output and explain why a message with the signature is accepted or rejected.

3. Prerequisites

Before starting, ensure the following are in place:

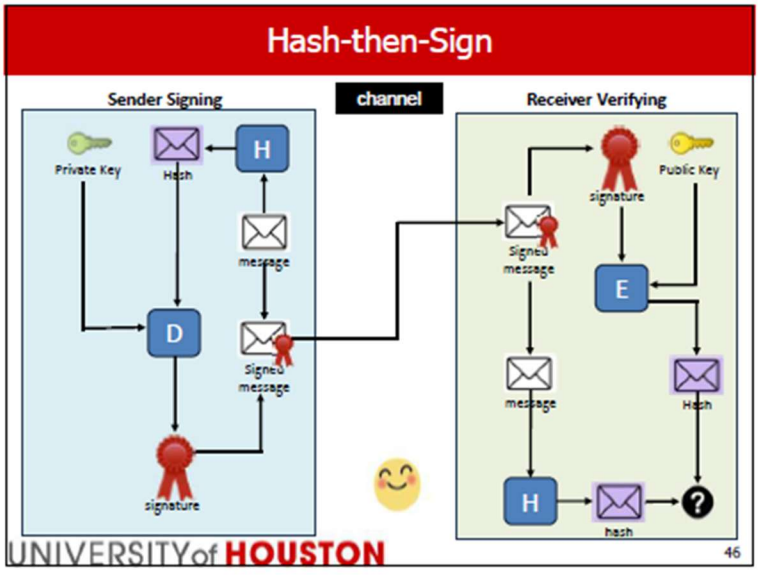
Software	Knowledge
Python 3.10+ (install package with: pip install cryptography)	Basic Python programming, strings/bytes, functions, and exception handling
PyCharm or VS Code IDE	Basic understanding of hashing, symmetric encryption, MAC, and public/private keys
The provided skeleton files: sender.py and receiver.py.	How does a digital signature provide authentication and integrity protection?

4. Background

Hash-then-Sign

This assignment is based on Chapter 8, Slide 46. For a message M, the sender computes a SHA-256 hash and signs it with the private key. The receiver uses the public key to verify the signature against the received message. If verification succeeds, the signature is accepted; if the message has been modified, verification fails.

Formula: h = H(M); S = Sign_private(M); verify with the public key



5. Files Provided

File	Purpose
Assignment4.docx	Assignment description
sender.py	Sender skeleton (signer): TODO blanks for SHA-256 hashing, RSA-PSS signing, and two test scenarios
receiver.py	Receiver skeleton (verifier): TODO blanks for SHA-256 hashing, RSA-PSS signature verification, and TCP receive loop

6. Tasks

Digital Signature / Hash-then-Sign (100 pts)

1. Open sender.py and receiver.py and fill in every TODO blank in both files.
2. Complete SHA-256 hash computation.
3. Complete the signing function using RSA-PSS with SHA-256 and the verification function using the public key.
4. In sender.py, complete the sender() function so that it converts the message to bytes, computes the SHA-256 hash, signs the message, prints the hash and signature, and transmits the message and signature to the server over TCP.
5. In receiver.py, complete the receiver() function so it receives the message and signature over TCP, computes the SHA-256 hash of the received message, and reports whether the signature is accepted or rejected.
6. Start receiver.py, then run sender.py with Scenario 1 (valid signature). Show that the sender and receiver output the report: "Signature accepted."
7. Start receiver.py, then run sender.py with Scenario 2 (tampered message sent with the original signature). Show that the sender and receiver output the report: "Signature rejected."
8. Write a short explanation of why the public key can verify the signature and why the modified message fails verification.

7. Deliverables (100 pts total)

Deliverable	Points
Completed sender.py and receiver.py with all TODO items filled in and code that runs correctly	50
Screenshot or captured output for Scenario 1 and Scenario 2	30
Short written explanation for digital signature verification and rejection	20
Total	**100**

8. Hints & Quick Reference

- Do not implement SHA-256 from scratch. Use the cryptography library.
- Always start the receiver before the sender. Keep your terminal output readable: label the sender side, receiver side, and each scenario clearly so the TA can follow your results.

9. Submission

Submit via the Canvas course portal one ZIP file named:

Assignment4_<YourLastName_YourID>.zip

Your ZIP file must include:

- the completed sender.py and receiver.py files
- one PDF report containing your screenshots, output, and short answers

Academic Integrity: All code must be your own. You may discuss concepts, but you must not share or copy code. Violations will result in a grade of zero.

Good luck - and remember, every secure connection you make on the web uses concepts you are implementing here!