

**Department of Telecommunication and Networking**  
**Specialized in cybersecurity**

**Digital signature  
system**

Course : Cryptography  
Year3 | Term 1  
Instructed by : Mr Meas Sothearith  
Date of submission : 20/12/2025

Submitted by : Huon Sreynuth

# Table of Contents

## 1. Introduction

- 1.1 Overview of the Project
- 1.2 Problem Statement
- 1.3 Motivation
- 1.4 Related Cryptographic Concepts

## 2. System Architecture

- 2.1 Overall Workflow
- 2.2 Key Generation Process
- 2.3 File Signing Process
- 2.4 Signature Verification Process
- 2.5 System Flowcharts & Diagrams

## 3. Implementation Details

- 3.1 Technologies and Libraries Used
- 3.2 Project Structure
- 3.3 Key Functions (from *signer.py*)
- 3.4 GUI Implementation (from *gui.py*)
- 3.5 Security Practices Followed

## 4. Usage Guide

- 4.1 Installation & Setup
- 4.2 Running the Program
- 4.3 Generating Keys
- 4.4 Signing Files
- 4.5 Verifying Signatures
- 4.6 Expected Outputs & Examples

## 5. Conclusion & Future Work

- 5.1 Summary
- 5.2 Future work

## 6. References

# 1. Introduction

## 1.1 Overview of the Project

This project implements a **Digital Signature System** using Python, Tkinter, and the *cryptography* library. The system allows users to:

- Generate RSA public/private keys
- Sign any file
- Verify signed files
- View SHA-256 hashes
- Export verification reports (JSON)
- Use a clean and beginner-friendly GUI

The goal is to help users understand how digital signatures ensure the authenticity and integrity of documents.

## 1.2 Problem Statement

In digital environments, files can be altered, forged, or intercepted. Without cryptographic protection:

- Anyone can modify a document
- The receiver cannot know the real author
- It becomes impossible to prove integrity

This project solves that problem through RSA signatures.

## 1.3 Motivation

Digital signatures are used everywhere banking, mobile payments, software updates, e-government services, legal documents, and more.

Building a functional signature system helps students understand:

- Cryptography in real systems
- Key generation
- Hashing
- Signing and verifying files
- How public-key security prevents tampering

## 1.4 Related Cryptographic Concepts

The system uses several concepts:

- **RSA Asymmetric Cryptography**
- **Public Key** (verification)
- **Private Key** (signing)
- **SHA-256 hashing**
- **RSA-PSS padding** (modern secure padding)
- **Authentication, Integrity, Non-repudiation**

## 2. System Architecture

The system consists of three major components:

1. **Key Generation**
2. **File Signing**
3. **Signature Verification**

All operations are controlled through a **Tkinter GUI**.

### 2.1 Overall Workflow

1. User generates RSA key pair
2. User selects a file to sign
3. System loads private key
4. Computes SHA-256 hash
5. Creates RSA-PSS digital signature
6. Saves `.sig` signature file
7. User selects file + public key + signature
8. System verifies signature
9. JSON report is generated

This ensures:

- Integrity

- Authentication
- Non-repudiation

## 2.2 Key Generation Process

Handled by `generate_keys()` in *signer.py*.

### Steps

- Create RSA private key (2048/4096 bits)
- Extract public key
- Save as:
  - `private_key.pem`
  - `public_key.pem`
- Display:
  - Modulus bit size
  - Public exponent

### Security Features

- Strong RSA keys
- Modern padding (RSA-PSS)

## 2.3 File Signing Process

Triggered from the GUI using `gui_sign_file()`.

### Steps

- Select file
- Load private key
- Read file bytes
- Compute SHA-256
- Generate RSA-PSS signature
- Save `.sig`
- Display SHA-256 hash

### Purpose

- Confirms sender identity
- Ensures file is unchanged

## 2.4 Signature Verification Process

Run using `gui_verify_file()`.

### Steps

- Load file
- Load public key

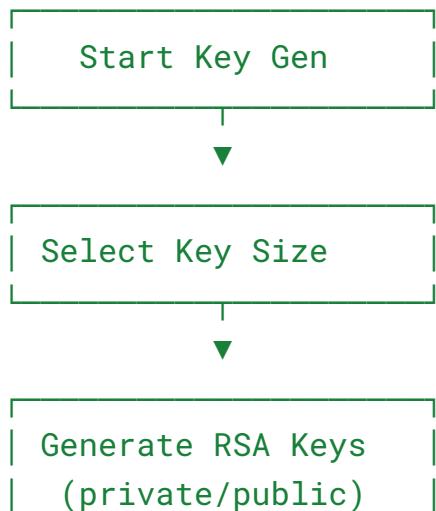
- Load `.sig`
- Recompute SHA-256
- Validate digital signature
- Save verification report (JSON)

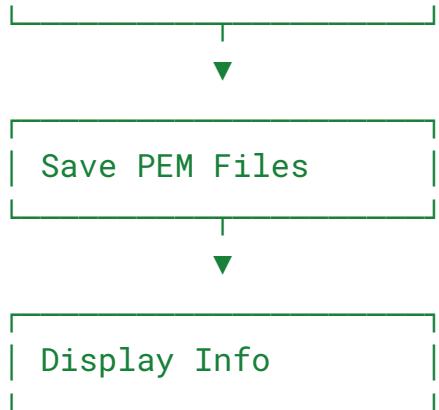
### Report Contains

- File name
- Public key used
- Signature VALID / INVALID
- SHA-256 hash
- Timestamp

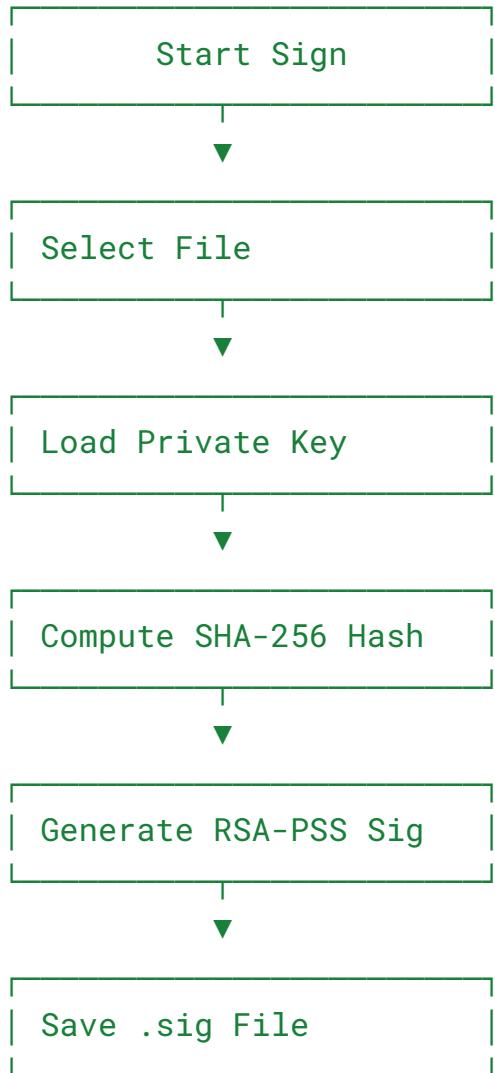
## 2.5 System Flowcharts & Diagrams

### Key Generation

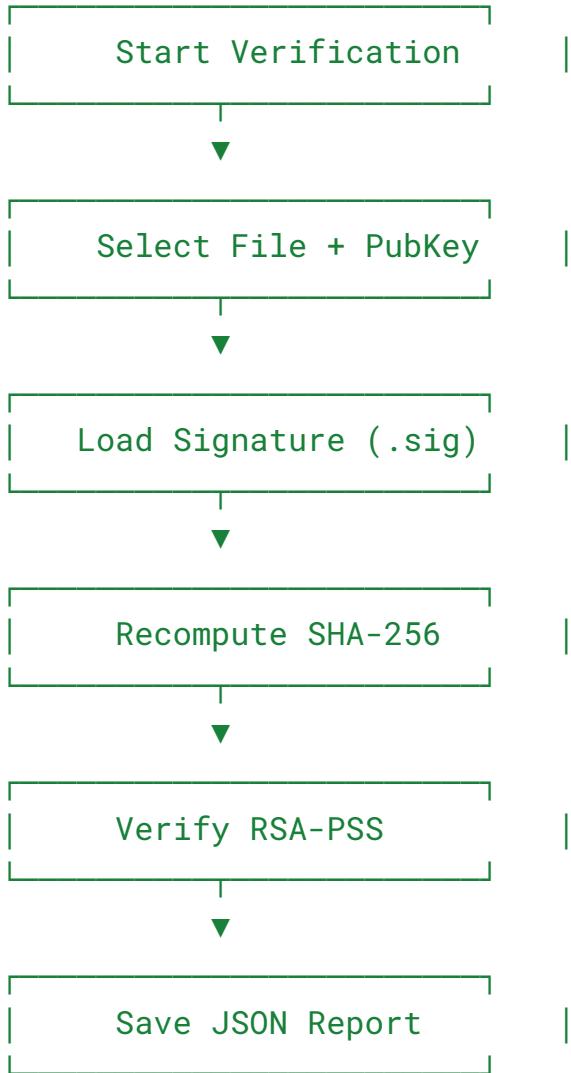




## File Signing



## Signature Verification



## 3. Implementation Details

### 3.1 Technologies and Libraries Used

Component	Technology
-----------	------------

Language	Python
GUI	Tkinter
Crypto	cryptography library
Hashing	hashlib (SHA-256)
Reports	JSON, datetime

## 3.2 Project Structure

```
digital_signature_system/
├── src/
│   ├── signer.py
│   └── gui.py
└── report/
    └── CRYPTREPORT.pdf
```

## 3.3 Key Functions (signer.py)

### a. generate\_keys()

- Creates RSA private/public key pair
- Saves keys in PEM format
- Returns modulus size and exponent

### b. sign\_file()

- Loads private key
- Reads file bytes
- Computes SHA-256 hash
- Generates RSA-PSS signature
- Saves `.sig` file

#### c. `verify_file()`

- Loads public key
- Loads file and signature
- Recomputes hash
- Validates signature
- Returns VALID / INVALID

#### d. `export_report()`

Saves:

- file name
- signature status
- pubkey used
- timestamp
- SHA-256 hash

## 3.4 GUI Implementation (gui.py)

### Major Features

- Pink GUI (your request)
- File browsing buttons
- Buttons for Sign / Verify / Generate Keys
- SHA-256 hash display
- Modulus/exponent display

### Main GUI Functions

- `gui_generate_keys()`
- `gui_sign_file()`
- `gui_verify_file()`
- `select_file()`

## 3.5 Security Practices Followed

- RSA-2048 and RSA-4096 supported
- RSA-PSS padding (modern + secure)

- SHA-256 hashing
- Clear exceptions for tampering
- JSON reports for traceability

## 4. Usage Guide

### 4.1 Installation & Setup

Install cryptography:

```
pip install cryptography
```

Run GUI:

```
python gui.py
```

### 4.2 Running the Program

The GUI opens directly—no CLI needed.

### 4.3 Generating Keys

1. Click **Generate Keys**
2. Choose key size
3. Save `private_key.pem`
4. Save `public_key.pem`
5. Modulus + exponent appear on screen

## 4.4 Signing Files

1. Choose file
2. Load private key
3. Sign
4. Save `.sig`
5. SHA-256 hash appears

## 4.5 Verifying Signatures

1. Choose file
2. Choose `public_key.pem`
3. Choose `.sig`
4. Output:
  - VALID
  - INVALID
5. Save report as JSON

## 4.6 Expected Outputs

### Signature Example

Signature saved to: C:/Users/Desktop/file.sig (e.g.)

## **Verification Example**

Signature is VALID

SHA-256: 9ac4f2...

## **JSON Report**

```
{  
  "file": "document.pdf",  
  "status": "VALID",  
  "public_key": "public_key.pem",  
  "sha256": "e3b0c442...",  
  "timestamp": "2025-12-06T14:22:10"  
}
```

# **5. Conclusion & Future Work**

## **5.1 Summary**

The system successfully performs:

- RSA key generation
- File signing
- Signature verification
- Integrity and authenticity checks
- GUI-based operations
- JSON report creation

## **5.3 Future work**

- Add X.509 certificate support
- Add HMAC mode
- Add file encryption + signing
- Add Docker deployment
- Add cloud key management

## 6. References

- Python Cryptography Documentation
- NIST Digital Signature Standard (FIPS 186-4)
- RFC 3447: RSA Cryptography Standard
- Tkinter Official Documentation
- Stallings, "Cryptography and Network Security"