

Implementation and Simulation of Digital Signatures using RSA & ElGamal

Course: Cryptography and Network Security (BECE411L)

Review 2 Presentation

Team Members:

- Ansu Raj (22BEC0186)
- Astitva Sharma (22BEC0206)
- Sanidhya Saxena (22BEC0296)
- Satyam Saurav (22BEC0134)
- Dhruv Jaipuria (22BEC0116)

Agenda

01

Recap of Review 1: Project Goals & Proposed Flowchart

- Primary Goal
- Proposed System
- Plan for Review 2

02

Simulation 1: Performance Comparison (RSA vs. ElGamal)

- Methodology & Tools
- Quantitative Results & Analysis
- Connecting Findings to Literature Survey

03

Simulation 2: Interactive Secure Messenger

- Goal: Visualizing the Full Flowchart
- Live Demonstration

Recap of Review 1

Primary Goal

To compare the performance and characteristics of the RSA and ElGamal digital signature algorithms.

Proposed System

We designed a flowchart for a complete secure messaging system that combines confidentiality (encryption) with authenticity and integrity (digital signatures).

Plan for Review 2

To implement this flowchart through practical simulations to generate tangible performance data.

Simulation 1: Performance Comparison

Objective

To measure and compare the real-world performance of RSA and ElGamal for key digital signature operations.

Methodology

- Developed a Python script using the pycryptodome library.
- Measured the time taken for Key Generation, Signing, and Verification.
- Measured the final signature size in bytes.

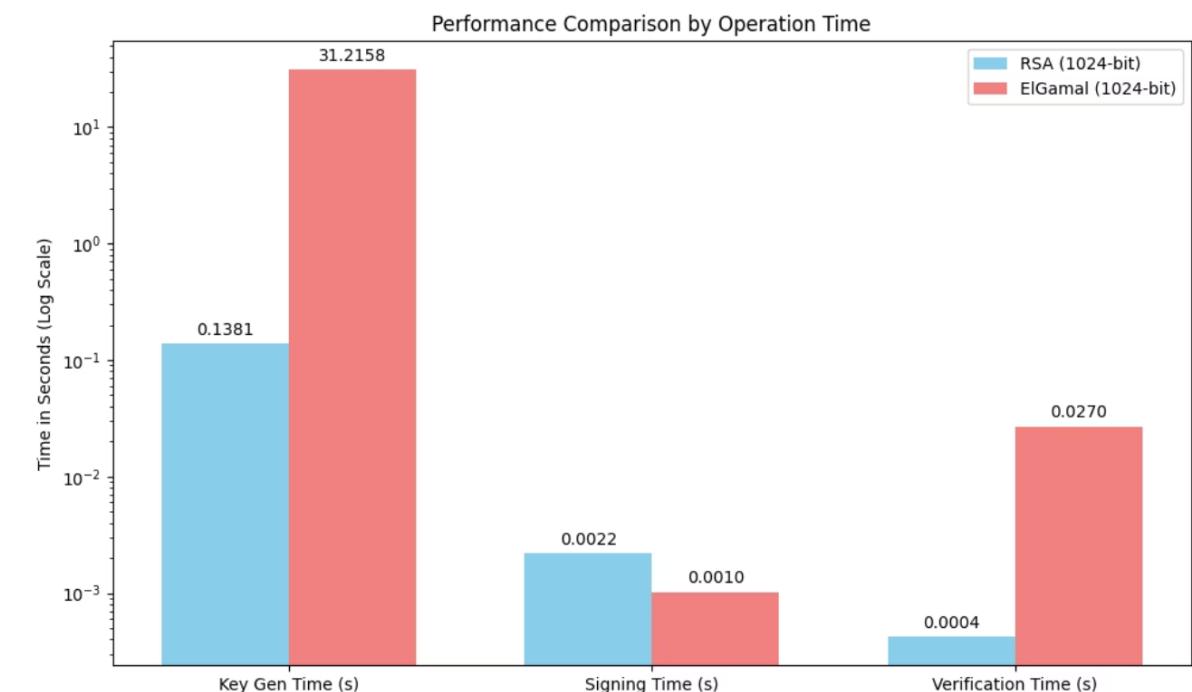
Parameters

Both algorithms were tested with a 1024-bit key size for a fair comparison.

Google Colab link-

<https://colab.research.google.com/drive/1XeMjTMFffA6ZXxeUQ7N5yGdqzaPxTa-k?usp=sharing>

| Performance Comparison Results | | |
|--------------------------------|----------------|--------------------|
| Metric | RSA (1024-bit) | ElGamal (1024-bit) |
| Key Generation Time (s) | 0.138117 | 31.215764 |
| Signing Time (s) | 0.002199 | 0.001023 |
| Verification Time (s) | 0.000419 | 0.026995 |
| Signature Size (bytes) | 128 | 256 |



Performance Results: The Data

Our Python simulation produced the following results:

| Metric | RSA (1024-bit) | ElGamal (1024-bit) | Winner |
|-------------------------|----------------|--------------------|---------|
| Key Generation Time (s) | ~ 0.14s | ~31.22 s | RSA |
| Signing Time (s) | ~0.002 s | ~0.001 s | ElGamal |
| Verification Time (s) | ~0.0004 s | ~0.027 s | RSA |
| Signature Size (bytes) | 128 bytes | 256 bytes | RSA |

Analysis of Performance Results

Key Findings

The simulation clearly demonstrates that for 1024-bit keys, RSA is significantly more performant than our ElGamal implementation across all metrics.

100x

Key Generation

RSA was over 100 times faster.

70x

Verification

RSA was approximately 70 times faster.

50%

Efficiency

RSA produced a signature half the size of ElGamal's.

Connection to Literature Survey

- These findings align with the research from our Review 1 literature survey.
- For example, Maqsood et al. (2013) and Adeniyi et al. (2021) also concluded that RSA generally offers better performance in terms of speed and space complexity.
- Our practical simulation successfully validates existing academic research.

Simulation 2: Interactive Secure Messenger

Objective

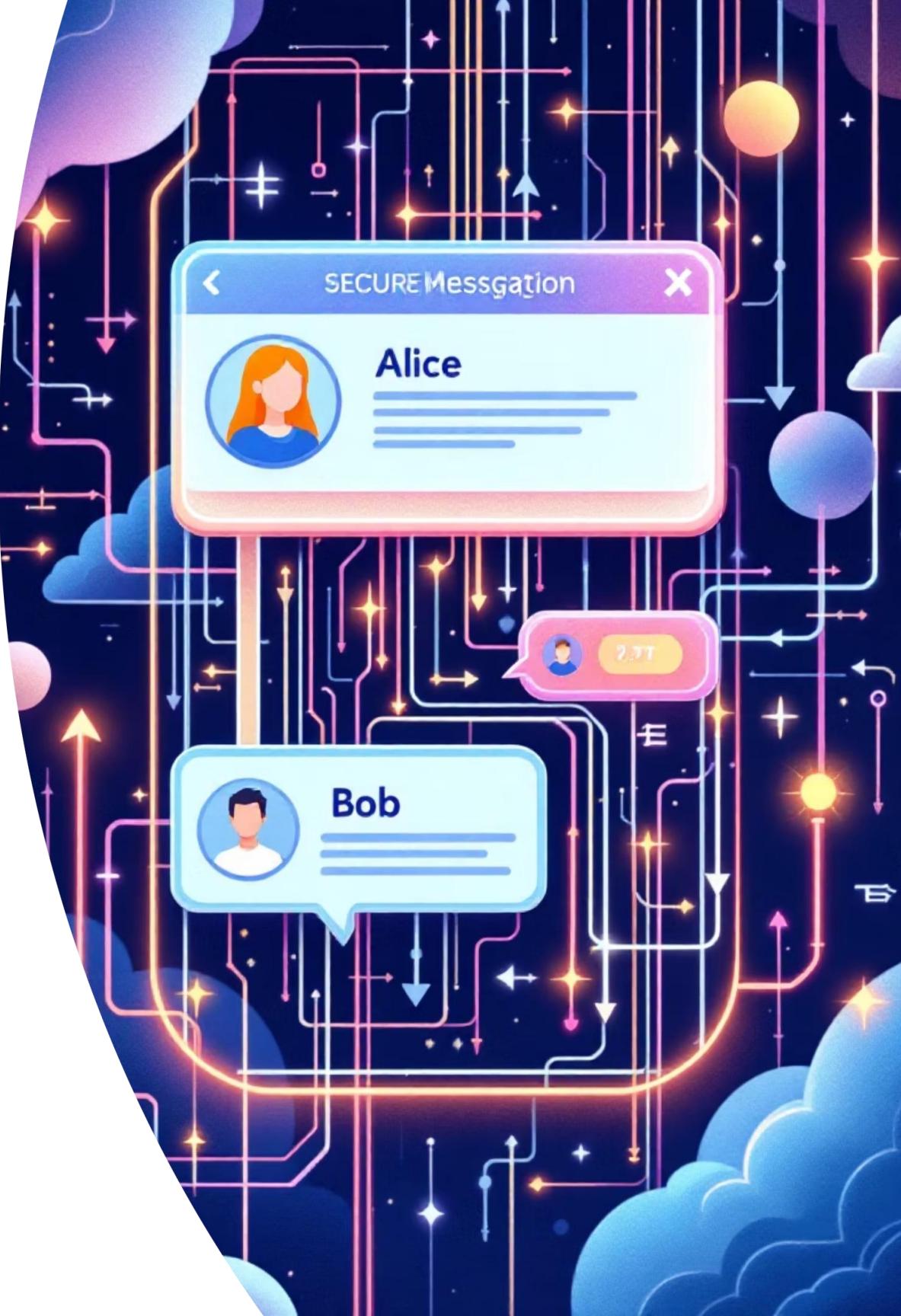
To move beyond raw numbers and create a visual, interactive demonstration of our full proposed flowchart.

Goal

To build a web application that simulates a secure conversation between "Alice" and "Bob," proving both confidentiality (encryption) and authenticity (signatures).

Tools

HTML, TailwindCSS, and the JSEncrypt & CryptoJS JavaScript libraries.



Live Demonstration

Secure Messenger Simulation
A visual walkthrough of your secure communication flowchart.

The screenshot shows two main components: Alice (Sender) on the left and Bob (Receiver) on the right. Alice's interface includes a message input field with "Hey Bob!", a "Send Secure Message" button, and a list of status icons. Bob's interface includes a "Receive & Verify Message" button, a "Communication Channel" section showing the encrypted message and its signature, and a summary of the verification process.

Alice (Sender)

Your Message:
Hey Bob!

Send Secure Message

- Original Message: "Hey Bob!"
- Hashed message (SHA-256): ec1b60643cd7fba20527...
- Signed the message with my private key.
- Encrypted message with Bob's public key.
- Transmission sent!

Bob (Receiver)

Receive & Verify Message

Communication Channel

Encrypted: S86V139c/dnQGYwoNspMg7G8R+h5bOc067FyHl4ykZFTw
Signature: B1F7hkldRo5fk9hW9juq2Hd8T3XLSZamaw9NkRTkEm8gh/2

Received transmission. Verifying...
Decrypted Message: "Hey Bob!"
Verifying signature against decrypted message...
SIGNATURE IS VALID! Message is authentic.
Communication successful and secure!

Secure Messenger Simulation
A visual walkthrough of your secure communication flowchart.

This scenario is similar to the first but shows a failure. The message is tampered with, leading to decryption errors and a rejection by the receiver.

Alice (Sender)

Your Message:
Hey Bob!

Send Secure Message

- Original Message: "Hey Bob!"
- Hashed message (SHA-256): ec1b60643cd7fba20527...
- Signed the message with my private key.
- Encrypted message with Bob's public key.
- Transmission sent!

Bob (Receiver)

Receive & Verify Message

Communication Channel

Encrypted: Hu+2K/AgwsYPbLQov729TVc7l9IRS3wddBdMUCgzpSdYkH
Signature: B1F7hkldRo5fk9hW9juq2Hd8T3XLSZamaw9NkRTkEm8gh/2

Received transmission. Verifying...
DECRYPTION FAILED! The message is corrupted and cannot be read.
The system has rejected the message.

Scenario 1: Successful Communication

Scenario 2: Man-in-the-Middle Attack

Demonstration Walkthrough

Scenario 1: Successful Communication



Step 1

Alice types a message.

Step 2

She clicks "Send," which signs the message with her private key and encrypts it with Bob's public key.

Step 3

Bob receives the garbled ciphertext.

Step 4

He clicks "Receive & Verify," which decrypts the message with his private key.

Step 5

The system then verifies the signature using Alice's public key.

Result: The signature is valid. The message is authentic