Farukh Saidmuratov

CSCI 4230

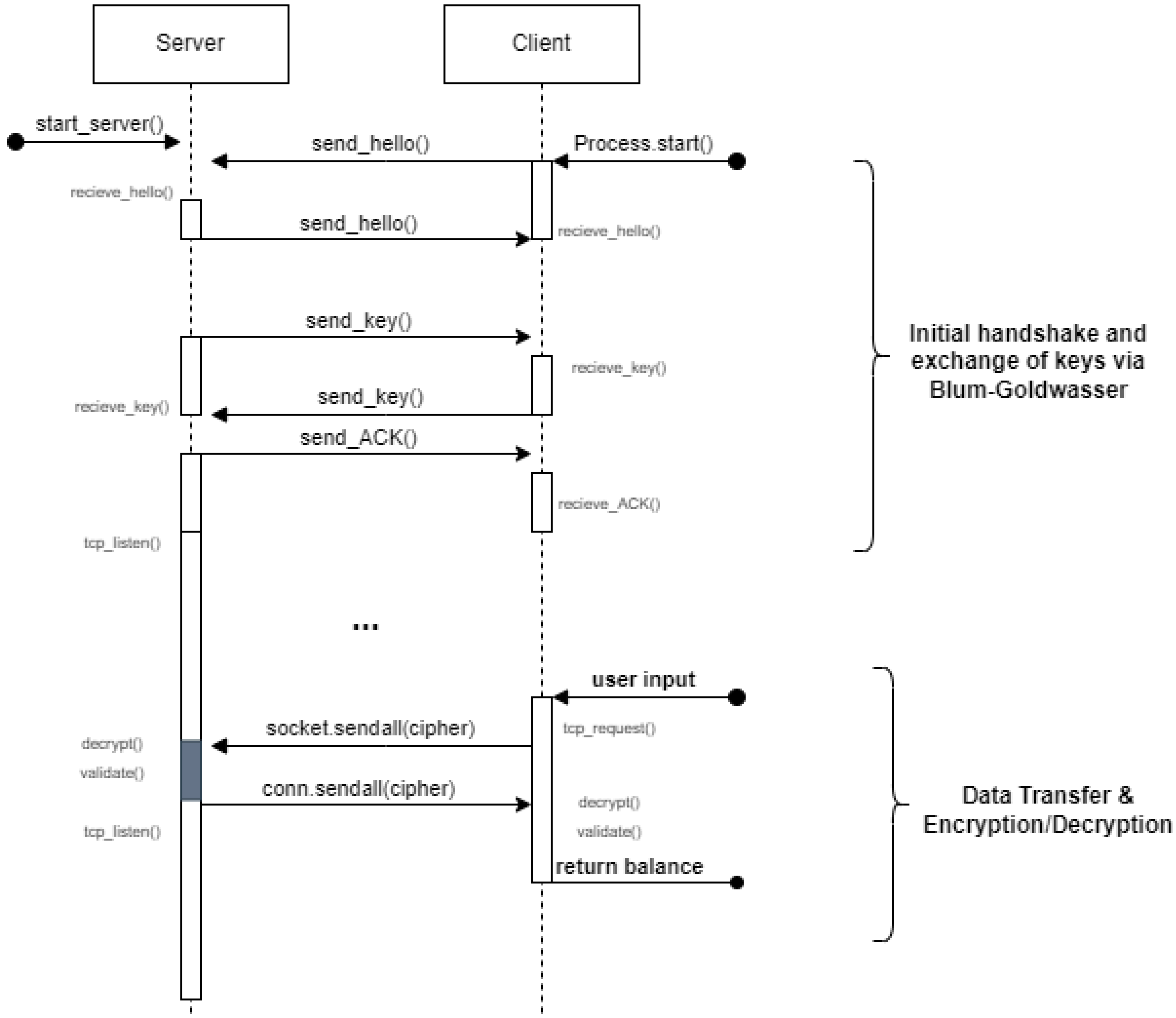
April 14th, 2024

ATM Final Project

This project is a CLI app, which describes a comprehensive crypto-system and Server/Client TCP communication between two objects that represent a bank(**Server)** and a bank account(**Client)**. Upon start, **Server** & **Client** go through the following steps:

1. **Server** and **Client** establish connection, and then share private keys using the **Blum-Goldwasser** scheme.
2. Once these keys are shared, **Server** then begins to listen for messages from **Client**.
3. **Client** sends user requests to **Server** which are encrypted via **Triple DES**.
4. **Triple DES** is again used to decrypt messages **Server-**side and the validity of MAC & timestamp are checked.
5. **Server** subsequently sends encrypted messages back to the **Client**.
6. Upon receiving the response from the **Server, Triple DES** is again used to decrypt messages **Client-**side and once again the validity of MAC & timestamp are checked.
7. This process continues until the user indicates that they would like to quit, or until a **timeout occurs due to an error in the validity of the message.**

This workflow ensures secure key sharing, as well as resistance against common passive & active attacks.This is further illustrated in the sequence diagram on the next page.



Within this report, I will describe the socket & model features that I worked on, as well as the encryption/decryption schemes that my teammates worked on, albeit in less detail. I will also describe the benefits as well as the pitfalls of our current implementation, and how we might address those pitfalls in future iterations of our project.

Usage & Set-Up

To begin running this project, a conda environment or a sufficient local environment is needed in order to guarantee Python version 3.11, and a few third party packages. These third party packages, and their usage is listed below:

- **bitarray**: Package to manipulate/create an array of bits

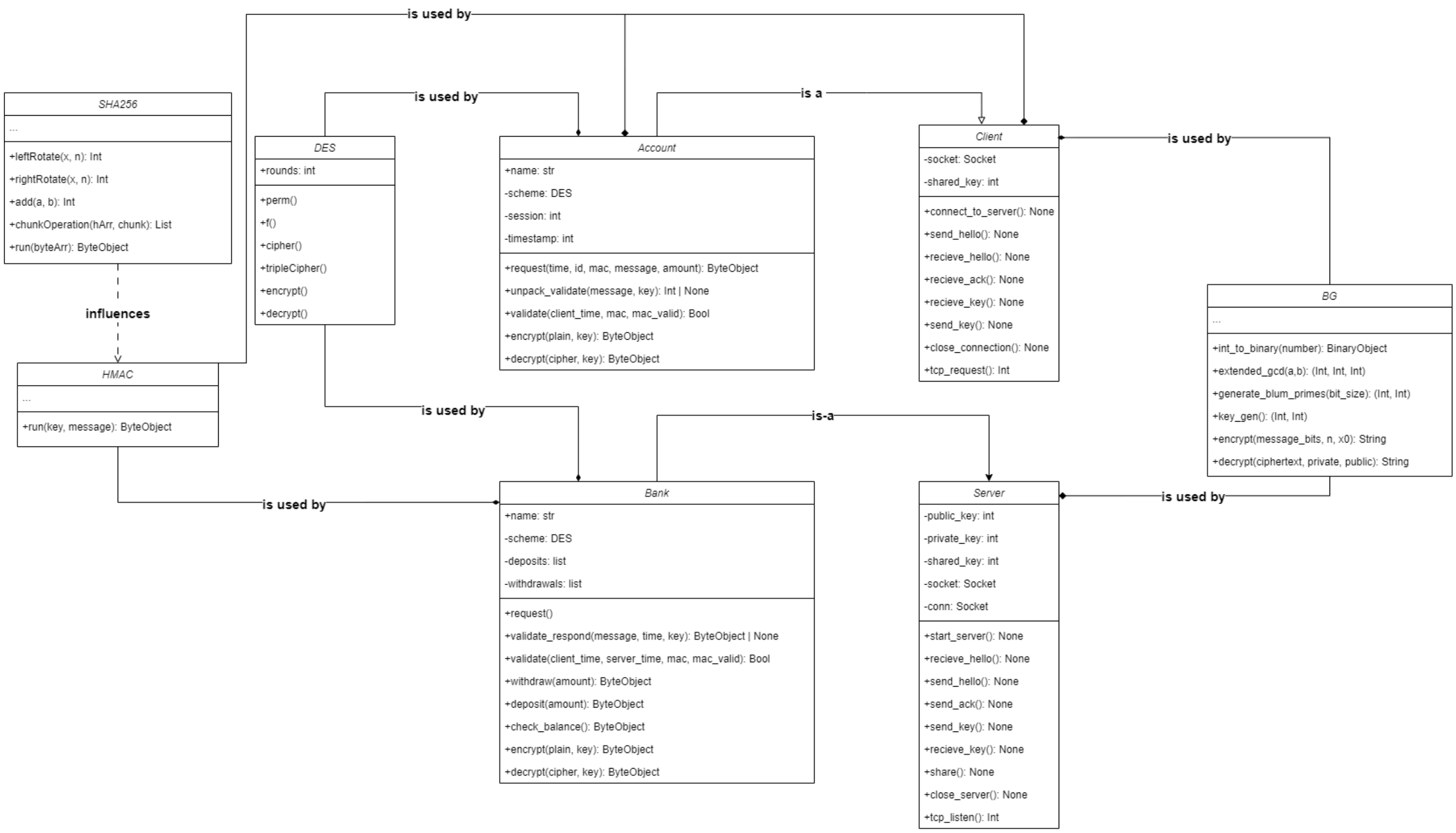
- **colorma**: Package to add color to terminal output
- **sympy**: Package to generate arbitrarily large primes

These packages could either be installed locally, or a conda environment could be created and activated to immediately be able to run this project. This project's entry point is the `run.py` file, which can be executed via `python -m run`. The following terminal header and prompt displays upon completion of handshake & key sharing.

Features

The classes of this project can be described in 3 discrete sections: **Model, Handshake & Key-Share, & Encryption Scheme**. The **Model** component contains the server/client code which handles all communication and usage of crypto schemes, whereas the **Handshake** & **Key-Share** describe the implementations of a handshake protocol via Blum-Goldwasser.

Encryption Schemes contain relevant Triple DES & HMAC generation via SHA256. This architecture is described in the Class UML diagram below:



Models

We abstract the operations between the Client & the Server using the **Bank, Account, Server,** and **Client** classes. Furthermore, all of these objects utilize the encryption schemes to securely share keys & send messages.

Bank	
Responsibilities	**Collaborators**
Interprets & validates client requests, modifies ledger, & sends updated balance to client. Encrypts & decrypts.	Inherited by **Server** to implement socket functionality. Uses **DES** to encrypt/decrypt messages.Uses **HMAC** for validation.

Account	
Responsibilities	**Collaborators**
Creates client requests, interprets & validates bank response, and returns updated balance. Encrypts & decrypts.	Inherited by **Client** to implement socket functionality. Uses **DES** to encrypt/decrypt messages.Uses **HMAC** for validation.

Client	
Responsibilities	**Collaborators**
Sends client requests, initiates handshake, handles errors.	Uses **BG** for key-sharing. Uses **HMAC** for MAC generation.

Server	
Responsibilities	**Collaborators**
Responds to handshake, listen and respond to client requests. Handles errors.	Uses **BG** for key-sharing. Uses **HMAC** for MAC generation.

Handshake & Key-Share

Blum-Goldwasser is used to share keys across channels.

Encryption Schemes

Triple DES is used to encrypt our messages. SHA256 & HMAC are used to encrypt our messages.

Benefits

- **Semantic Security During Key-Sharing:** As the BG algorithm is semantically secure via random generation of numbers…

- **Authentication:** We authenticate responses from both parties by checking the validity of the MAC as well as reasonable timestamps. This defends against man-in-the-middle and replay attacks.
- **Integrity Checking:** We also check for the integrity of the message itself, and only accept messages that entail one of the 3 idiomatic requests from the Client (*CHECK, WITHD, DEPOS*).
- **Errors & TimeOut:** Upon violation of the integrity of a message, we close the server and hang the child process, as opposed to giving back an error message which could be used to deduce the …

Pitfalls & Next Actions

- **Error Handling:** Error messages potentially reveal structure of code and therefore implementation.
- **Homomorphic Encryption:** Ensure that all stored information in the “ledger” is encrypted, while still permitting for calculation of the balance via homomorphic encryption.