E-Banking

Jason Pu (put@rpi.edu) Keran Wang (wangk16@rpi.edu)

SECTION ONE. INTRODUCTION

In today's digital age, the security of personal and financial information sharing is paramount. Banking systems are no exception, and ensuring their safety is critical to maintaining customer trust and preventing financial losses. This paper describes in detail the technical aspects of a secure banking system, including its server setup, network processing, secure channel setup, and user workflow. The system also employs various cryptographic techniques, including RSA, DES, SHA1, and HMAC, to ensure secure communication and data protection. In the following article, the bank will be referred to as the server, and the ATM will be referred to as the client.

SECTION TWO. INFRASTRUCTURE & NETWORK

I. Server Setup

At startup, the server configures its RSA keypairs (see section RSA for more details). Then the server configures its SuperSecureSocket (see section SuperSecureSocket for more details) to create a new socket object for IPV4 addresses, and this socket will use TCP protocol for streaming data. This newly created socket object will be bound to a specified port, where it will be listening and waiting for incoming connection requests.

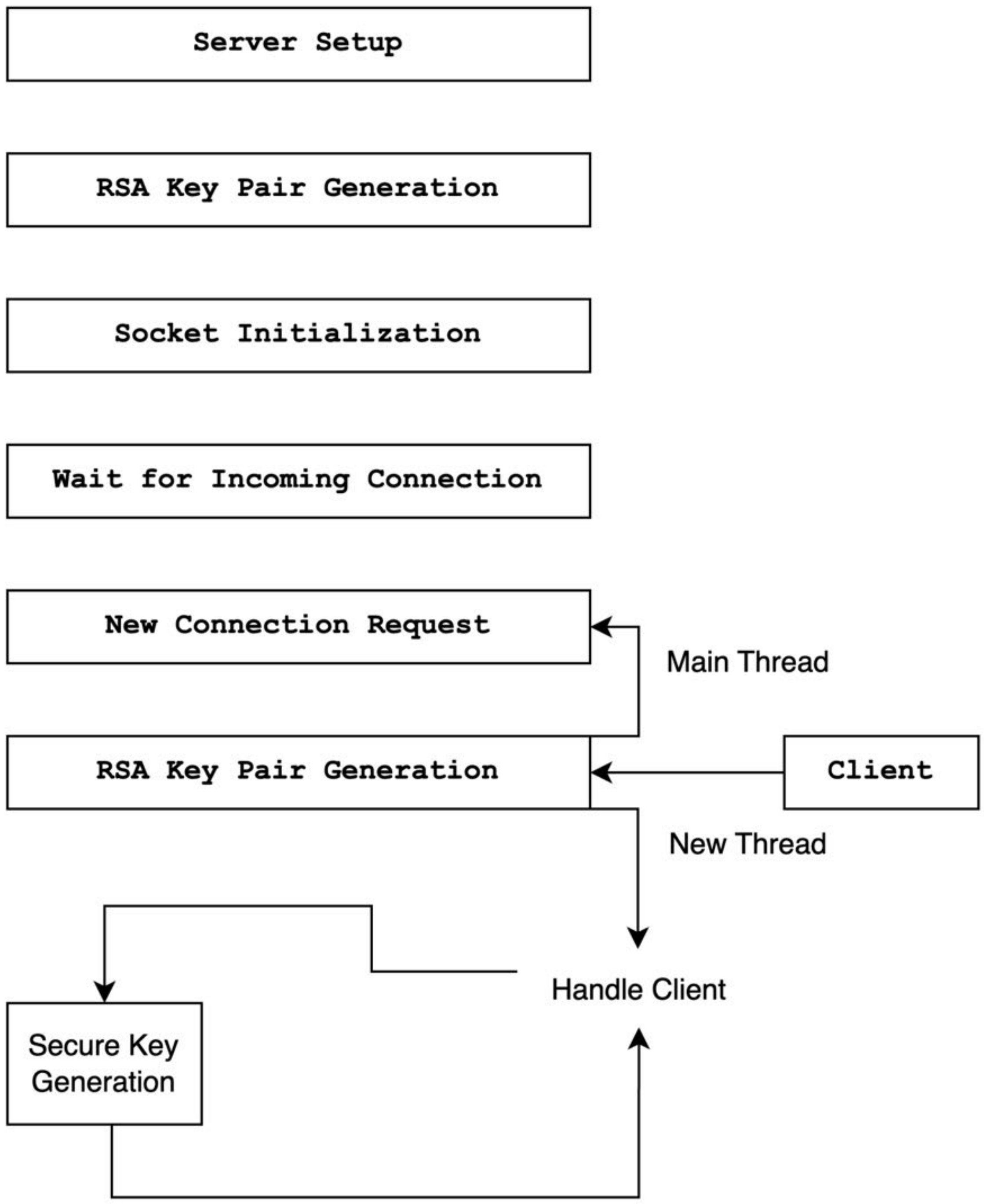


Figure 1. Server Setup Process

II. Handling Multiple Clients

The server is concurrent, and it can handle many clients at the same time. In its architecture, each client connection is handled by a separate thread, allowing multiple clients to be served simultaneously. The main thread of the server listens for incoming client requests, creates a new thread to handle each request, and then returns to listening for new incoming requests. When a thread is assigned to a client, it is responsible for handling all communication with that client until the client disconnects.

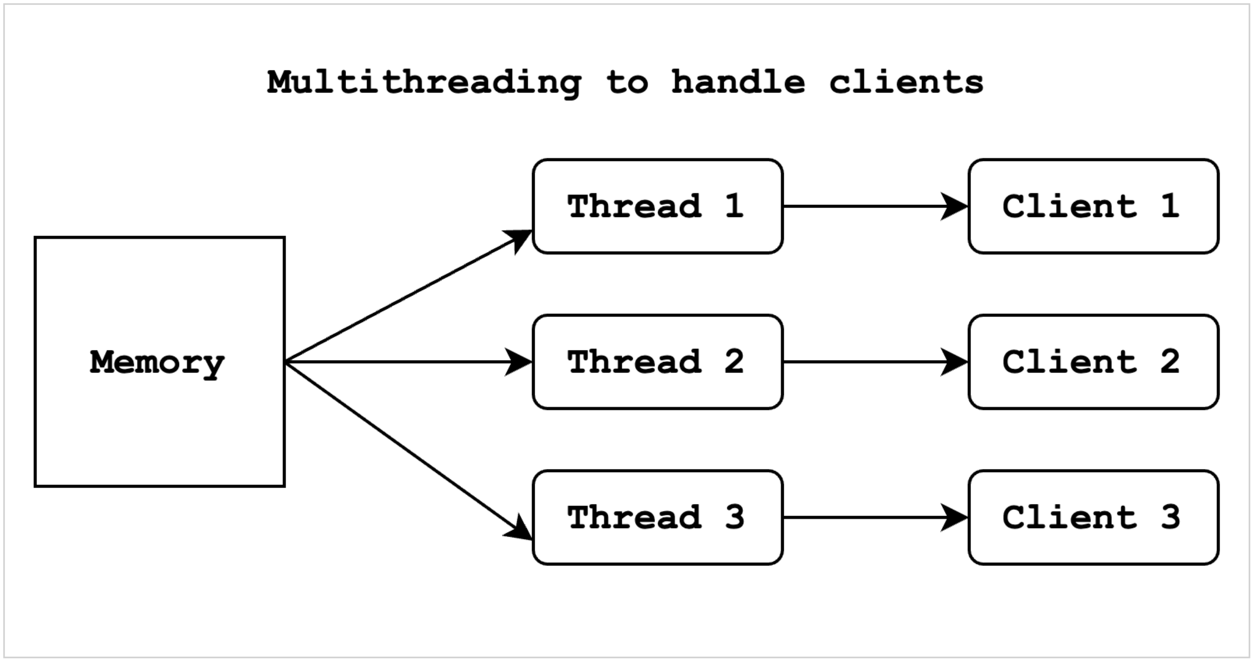


Figure 2. Multithreading to Handle Clients

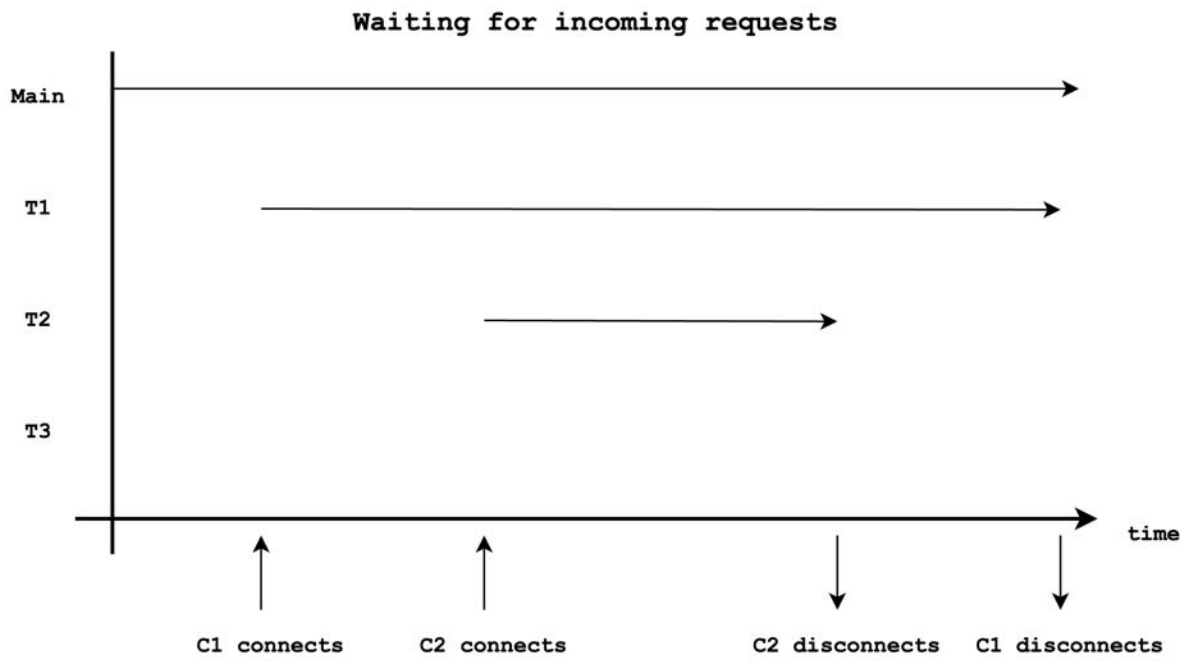


Figure 3. Main Thread Waiting for Incoming Requests

III. The SuperSecureSocket Wrapper

In general, a wrapper class is a class that encapsulates and provides a simplified or more specialized interface to another class or API. The SuperSecureSocket class (will be referred to as the SSS class for simplicity) is a wrapper class based on the python.socket package. It provides a layer of abstraction on the low-level socket implementation so that integration is easier in the process of development. More importantly, it adds several key functionalities.

The SSS wrapper keeps track of the last time any message is sent using this socket. With this functionality, the server is able to determine if any user has been inactive and terminate the current session.

The SSS wrapper provides an integration point for the encryption and decryption algorithms (See section Three for more details). Any data entering SSS is plain text. SSS encrypts the data so that it is secure in the process of transmitting. On the receiving end, SSS decrypts the received data and outputs the plaintext.

On top of encryption and decryption, the SSS wrapper also provides methods for validation (See section Three for more details). Before sending the data, the SSS puts a digital signature on top of it. And upon receiving the data, the SSS verifies the signature to make sure it has not been modified by adversaries.

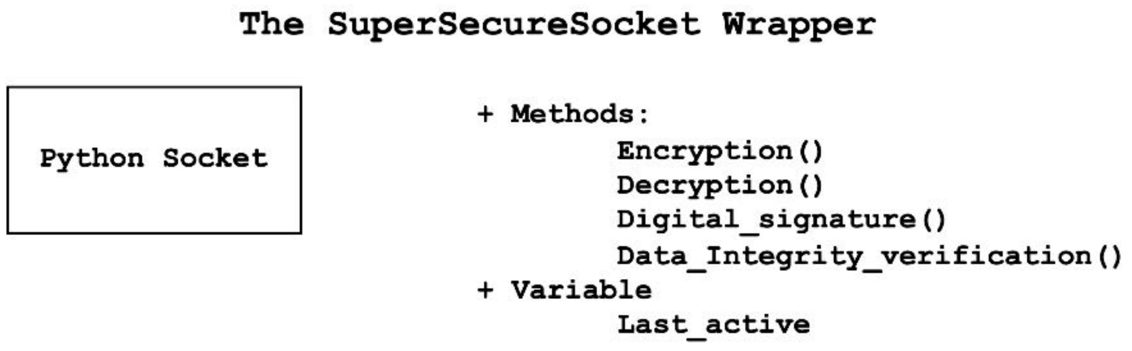


Figure 4. The SuperSecureSocketWrapper

IV. Secure Channel Setup

This section explains how the secure channel of communication between the server and the client is set up. First, the client reaches out to the server via a socket that is listening on a predefined port location. As a part of the TCP protocol handshake, the client sends an SYN (synchronize) packet to the server, indicating its intention to initiate a connection. The server responds with an SYN-ACK (synchronize-acknowledge) packet, acknowledging the client's request and indicating its own intention to establish a connection. The client sends an ACK (acknowledge) packet, confirming that it has received the server's response and is ready to begin communication. Then the server sends its public key to the client, which has been generated earlier. Then the client generates the session key. The client encrypts the session key using the public key from the server with RSA and sends the encrypted key to the server. Upon receiving the encrypted key, the server will decrypt the session key using its own private key. After this point, both parties have the symmetric session key, and all following messages will be encrypted and decrypted with it using the SSS wrapper.

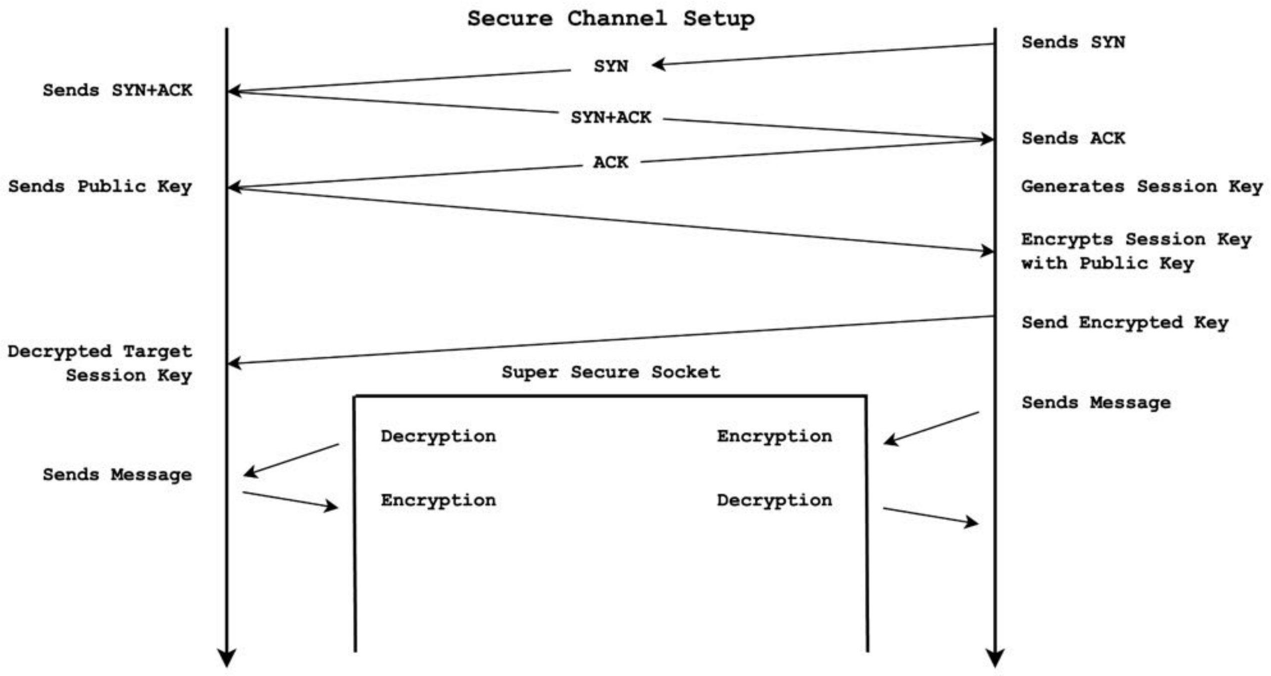


Figure 5. Secure Channel Setup Process

V. User Workflow

A user will need an account to access the bank. The user can either create an account or log in if the account already exists. When creating accounts, there are certain limitations as to the complexity of the password to ensure security. A single account cannot be logged in at two different clients(ATMs) at the same time. After successfully logging in, the user can perform actions including checking balance, depositing money, and withdrawing money. When withdrawing money, the amount cannot exceed what the user has in the balance. The user can also exist. If the user does not have operations for a certain amount of time, the user will be logged out due to inactivity.

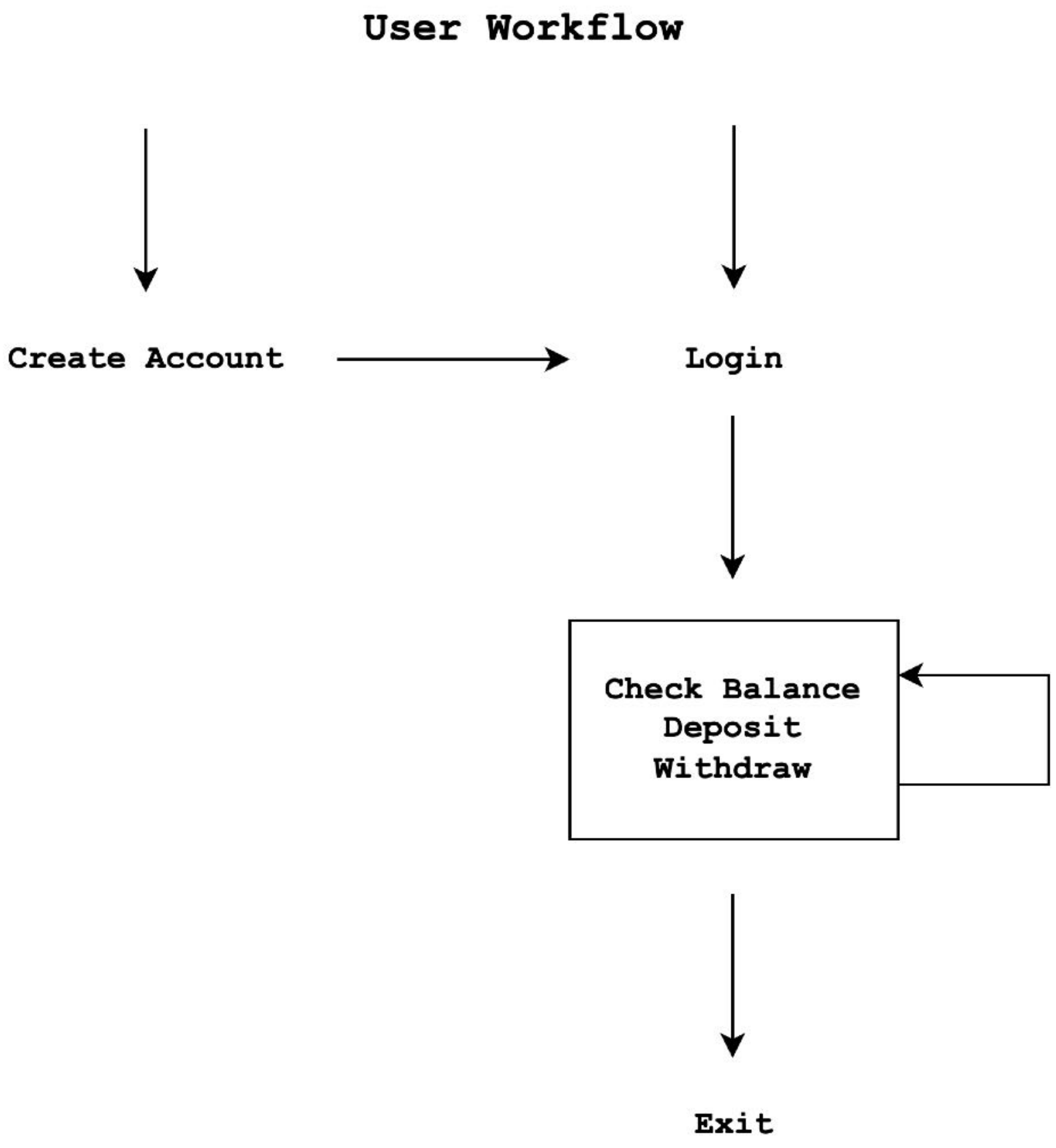


Figure 6. User Workflows

SECTION THREE. CIPHER

I. RSA

A. Description

The RSA algorithm works by using two large prime numbers to generate a public and private key pair. The server publishes the public key, while keeping the private key secret. To encrypt a message using the RSA, the sender uses the recipient's public key. Such cipher can only be decrypted using the recipient's private key.

B. Implementation

In our implementation, RSA is primarily used to encrypt the session key between the client and the server. For each character in the plaintext, a corresponding cipher is created. It applies the RSA encryption formula using the public key and the modulus. Specifically, it raises the Unicode code point of the character to the power of key and then takes the result modulo n. Combining each cipher character gives the cipher text.

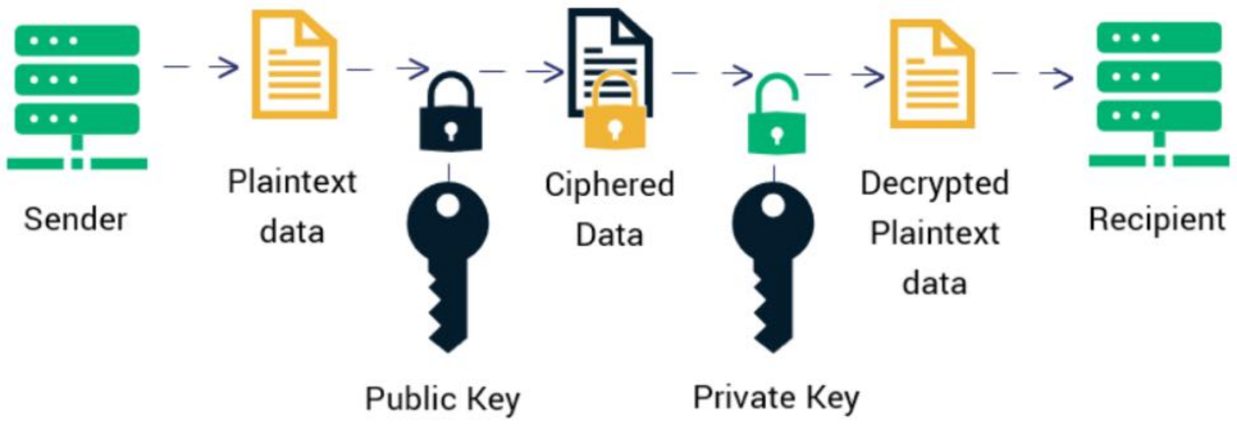


Figure 7. RSA Workflow
Source: https://sectigostore.com/blog/ecdsa-vs-rsa-everything-you-need-to-know/

II. Data Encryption Standard (DES)

A. Description

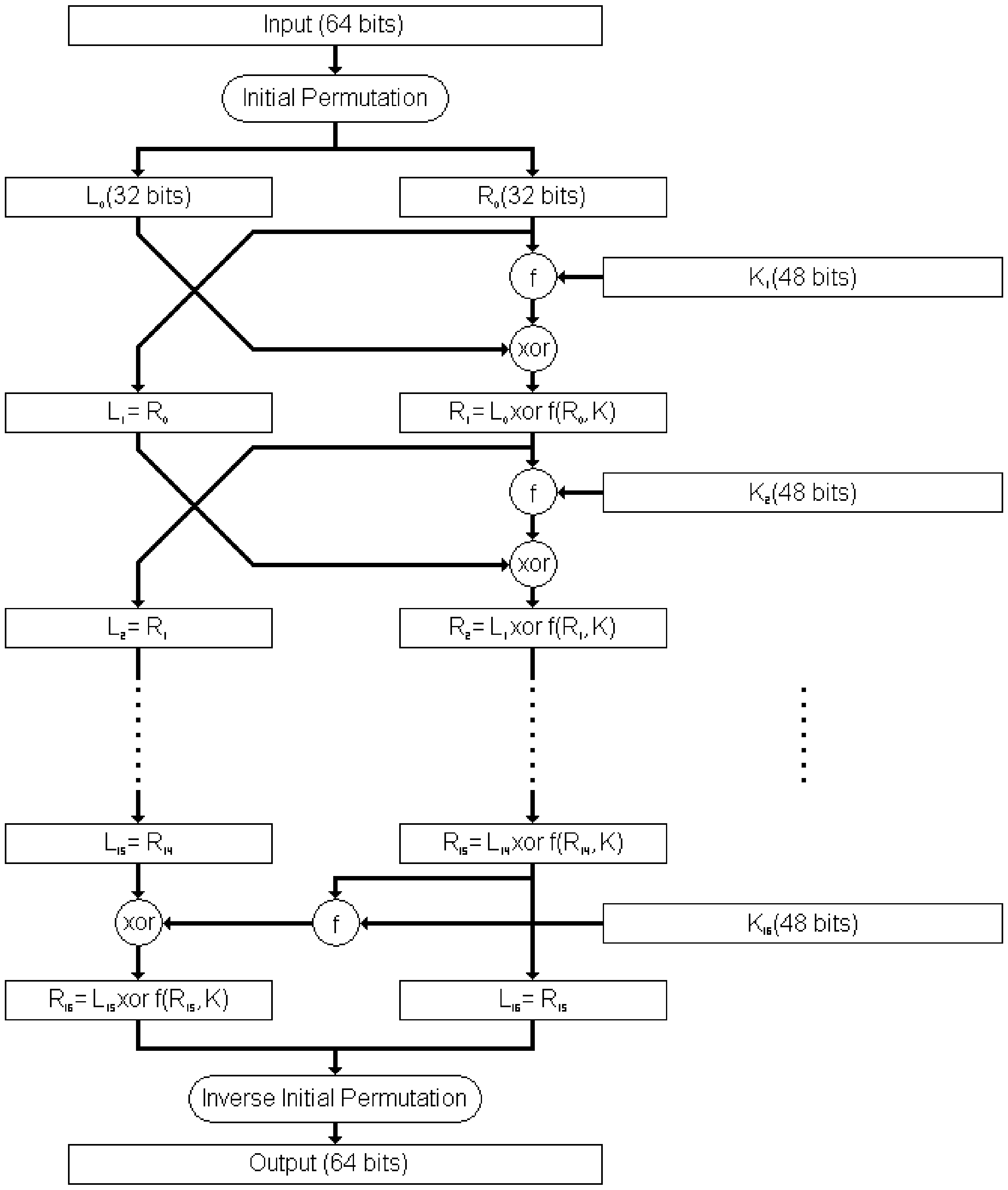


Figure 8. DES Implementation
Reference: https://manansingh.github.io/Cryptolab-Offline/c13-des-block.html

For encryption and decryption, we use DES, a symmetric key block cipher that uses a series of substitution-permutation operations to transform the plaintext into ciphertext. It was widely used before, and although it is no longer considered secure due to its small key size, we selected it partly because we want to leave room for the black hat part.

B. Implementation

We provide a simple implementation of the DES algorithm. First, we define some necessary static variables: SBOX, IP, P, PC1, PC2, FP, and EXP, which are predefined tables that represent the various transformations used during encryption and decryption. SBOX lists eight substitution boxes to perform nonlinear substitutions on the input data. IP is a permutation table that rearranges the bits of the input text fixedly at the beginning of the encryption process. P is a permutation table that rearranges the bits of the output of the SBOX before XORing with the left half of the data. We use PC1 and PC2 for key generation. FP is a permutation table that rearranges the bits of the output of the last round before outputting the final encrypted data. EXP is used to expand the right half of the input data to more bits.

Next, we define a class named DES which implements the DES algorithm for data encryption and decryption. The class has an init() method, which takes a key as input and initializes the key attribute of the class after encoding it in ASCII. The des() method is the main function that performs the DES algorithm on the input text. The input text is first split into two equal-length halves and subjected to an initial permutation using the initial permutation table (IP). The right half is then expanded and subjected to a series of substitutions and permutations for 16 rounds, depending on whether the mode is encrypted ("ENC") or decrypted ("DEC"). In each round, the right half is split into eight blocks, each subjected to substitution using the predefined substitution boxes (SBOX) and then permutation using the P permutation table. The result is XORed with the left half, and then we swap the left and right halves. Finally, the output is subjected to a final permutation using the final permutation table (FP). The proc() method is a helper function that takes the input text and mode as arguments and splits the input text into 8-bit blocks before calling the des() method on each block. The des() method output is concatenated and returned as the final encrypted or decrypted result.

III. Secure Hash Algorithm 1 (SHA1)

A. Description

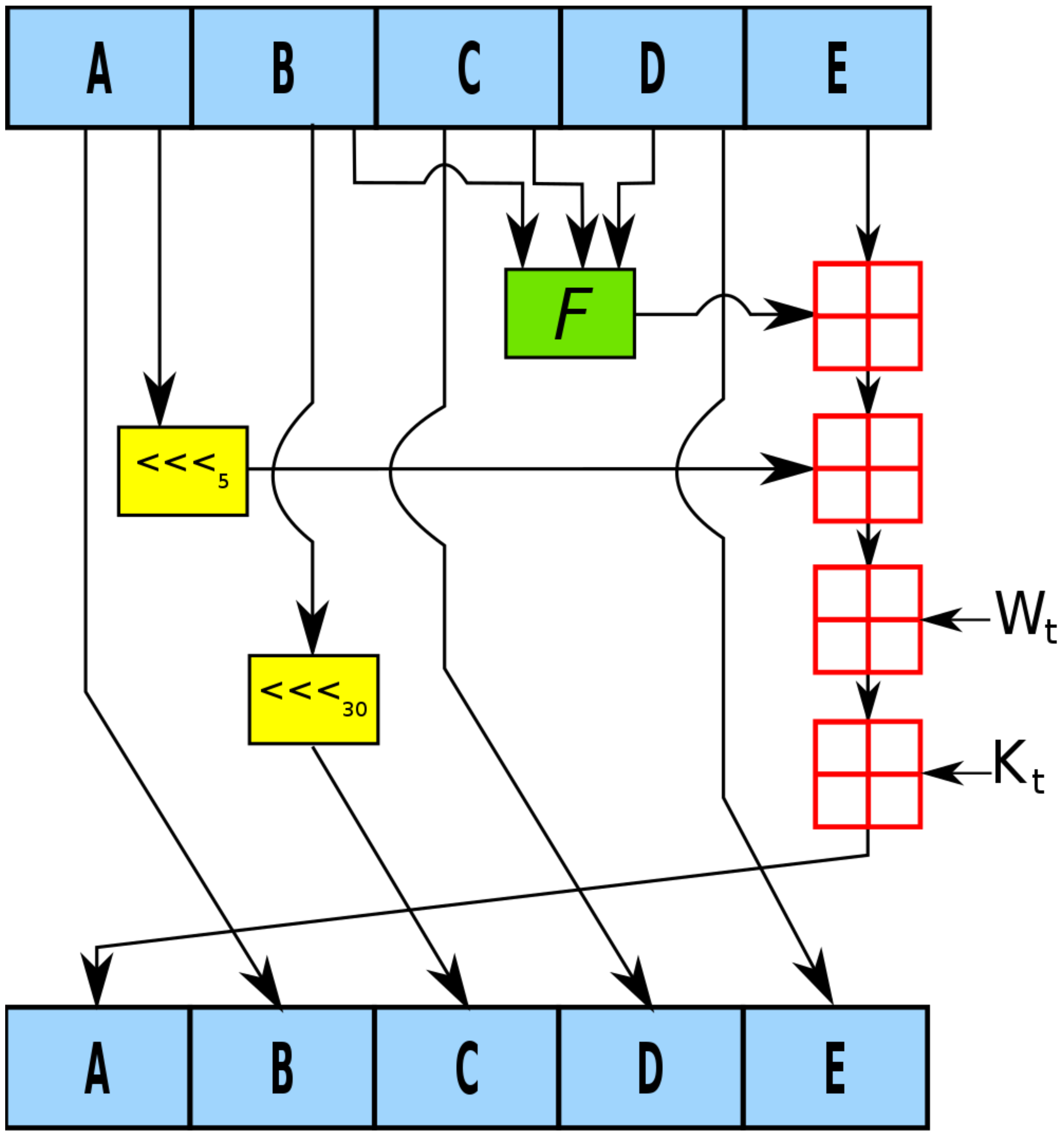


Figure 9. SHA-1 Implementation
Source: https://en.wikipedia.org/wiki/SHA-1

SHA1 is a widely used cryptographic hash function that produces a hash value, which can be used as digital signatures and be used in other security-related applications. The

SHA1 algorithm processes an input message through a series of mathematical operations to produce a fixed-length output, which represents the hash value.

B. Implementation

The SHA1 implementation is from Ben Wiederhak's implementation in pure Python. We choose this implementation because it does not require any additional external packages, and it's efficient and ready to use.

IV. Hash-based Message Authentication Code (HMAC)

A. Description

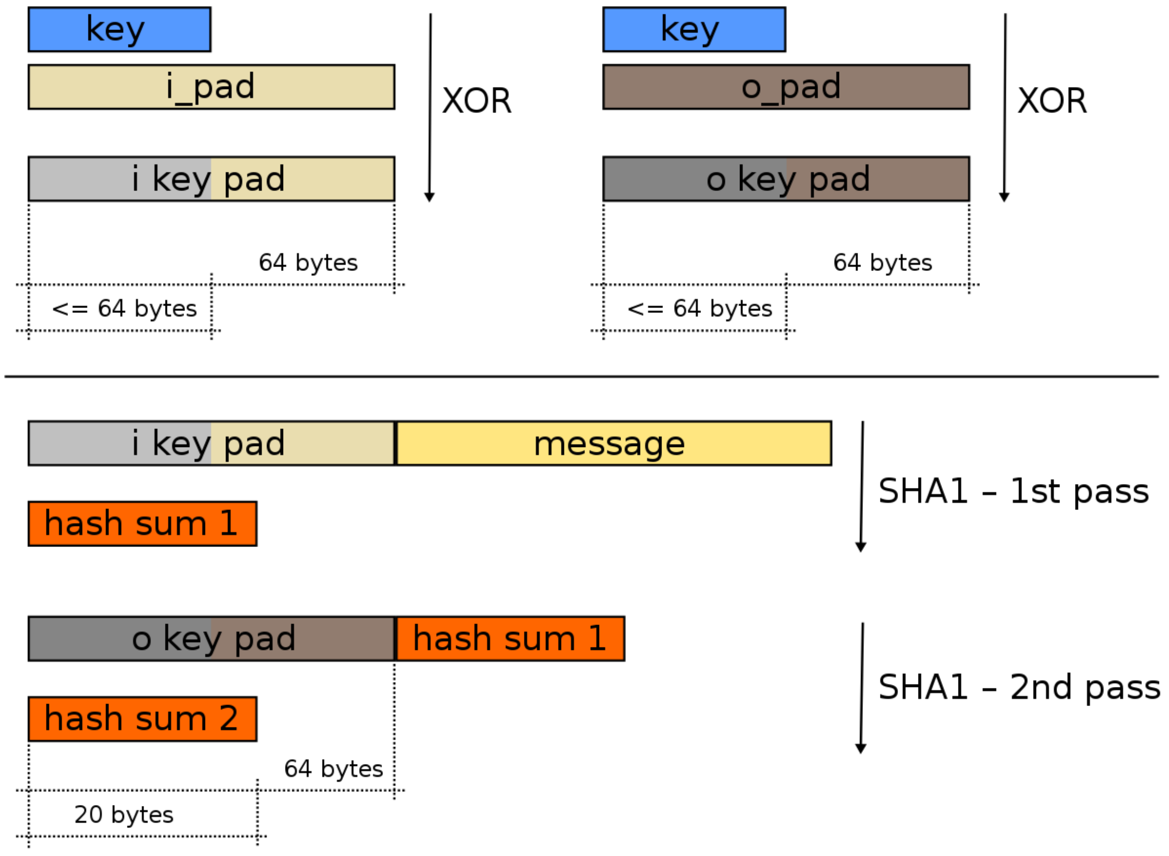


Figure 10. HMAC Implementation
Source: https://en.wikipedia.org/wiki/HMAC

We use HMAC, a mechanism that uses a secret key and a cryptographic hash function to verify the authenticity and integrity of a message. By appending an HMAC value to a message, the receiver can verify that the message came from the intended sender and was not interfered with during transmission. HMAC also provides a way to detect replay attacks, where an attacker attempts to resend a previously sent message to impersonate the sender.

B. Implementation

We first define a function called hmac_sha1 that takes a key and a message as input and pads it with zeros if the key is less than 64 bytes long. It then performs an XOR operation between the padded key and 0x36 bytes to generate the inner key and an XOR operation between the padded key and 0x5c bytes to generate the outer key. Then, it performs a hash operation on the inner key with the message using SHA-1 and a hash operation on the outer key with the resulting hash operation to generate the final HMAC.

We then define a DESHMAC class that uses HMAC for message authentication and DES for message encryption. It receives a key as input, initializes a DES object using that key, and generates an HMAC key using the SHA-1 hash function by calling the _preproc function. The encryption function takes a message as input -> generates an HMAC signature using the hmac_sha1 function -> concatenates the message and the signature -> converts the concatenated message to binary format using the bin function -> encrypts the binary message using the proc function of the DES object. The decryption function takes a cipher_text as input -> decrypts it using the proc function of the DES object -> converts the decrypted binary message to a string format -> separates the message and the signature -> verifies the authenticity of the message by comparing the signature with the newly generated HMAC signature -> returns the message if it is authentic, otherwise returns an empty byte string (b"").

SECTION THREE. EXPERIMENTS AND DISCUSSION

I. Preliminary

A. Programming Language

1. Python 3.9.12

B. Dependencies

1. Socket

   The Socket library is an essential part of the program. It is used to provide a set of methods for creating and using sockets. It is the underlying module that the SSS class is based on.

2. Threading

   The Threading library provides a way to run multiple threads within a single program. It allows for the concurrent execution of tasks while keeping everything lightweight. It helps the server to establish connections with many clients.

3. Decimal

   The Decimal library provides a way to perform decimal conversion and arithmetic operations.

4. Pickle

   The Pickle library provides a way to serialize and deserialize Python objects. Our implementation relies on it to convert Python objects into a stream of bytes so that they can be transmitted over the network. On the receiving end, it helps to reconstruct the object from serialized bytes from the socket.

5. Time

   The Python Time library provides functions to manipulate the time object. In our implementation, it is used to keep track of the time since the last activity in SSS.

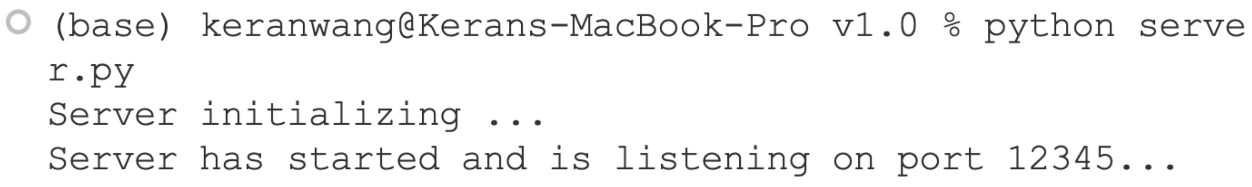
6. NumPy

   NumPy is a Python library used for working with arrays.

7. Random

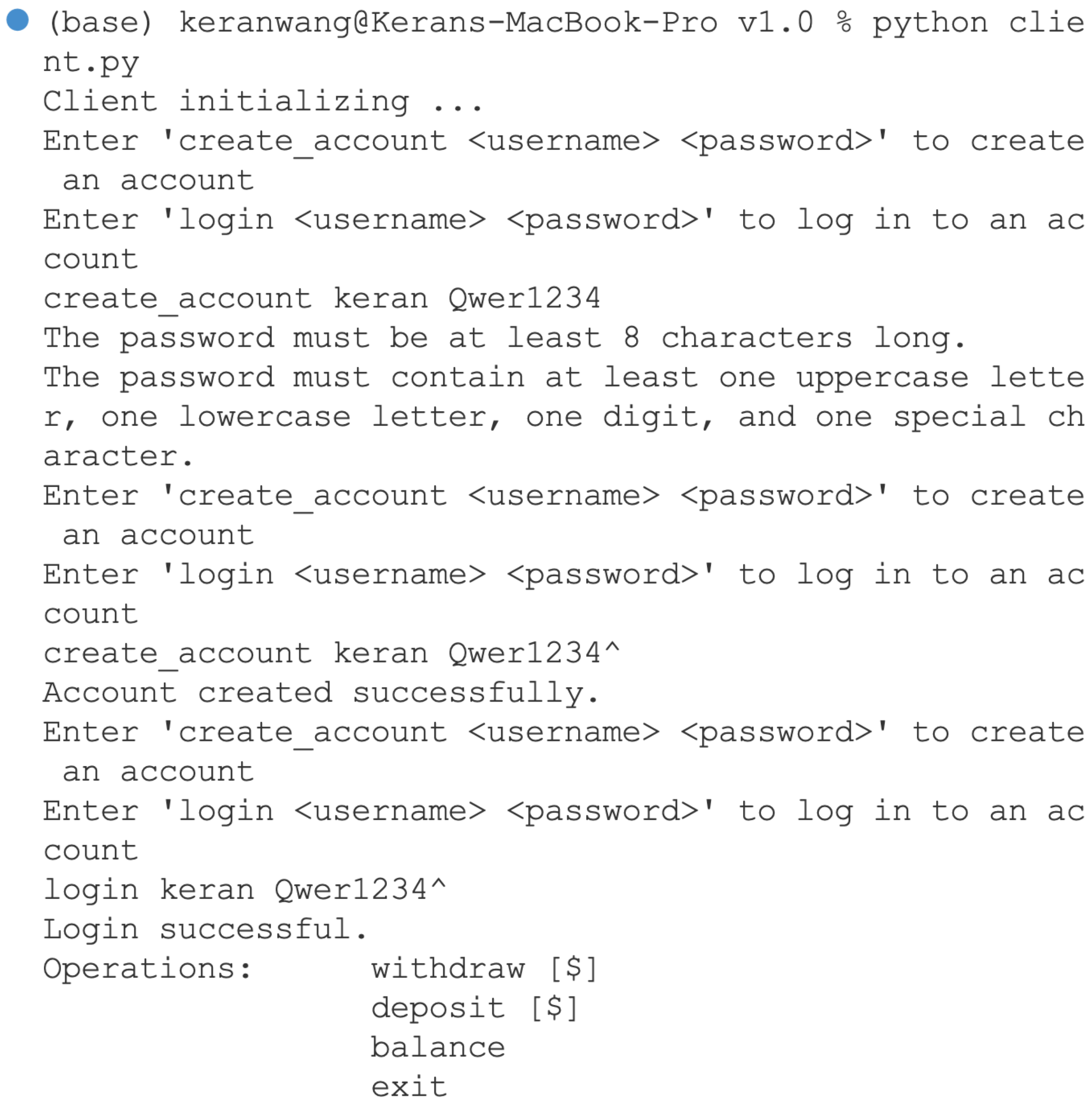
   The Random library can generate random numbers. It is used in various encryption algorithms in our implementation.

II. Experiments and Tests



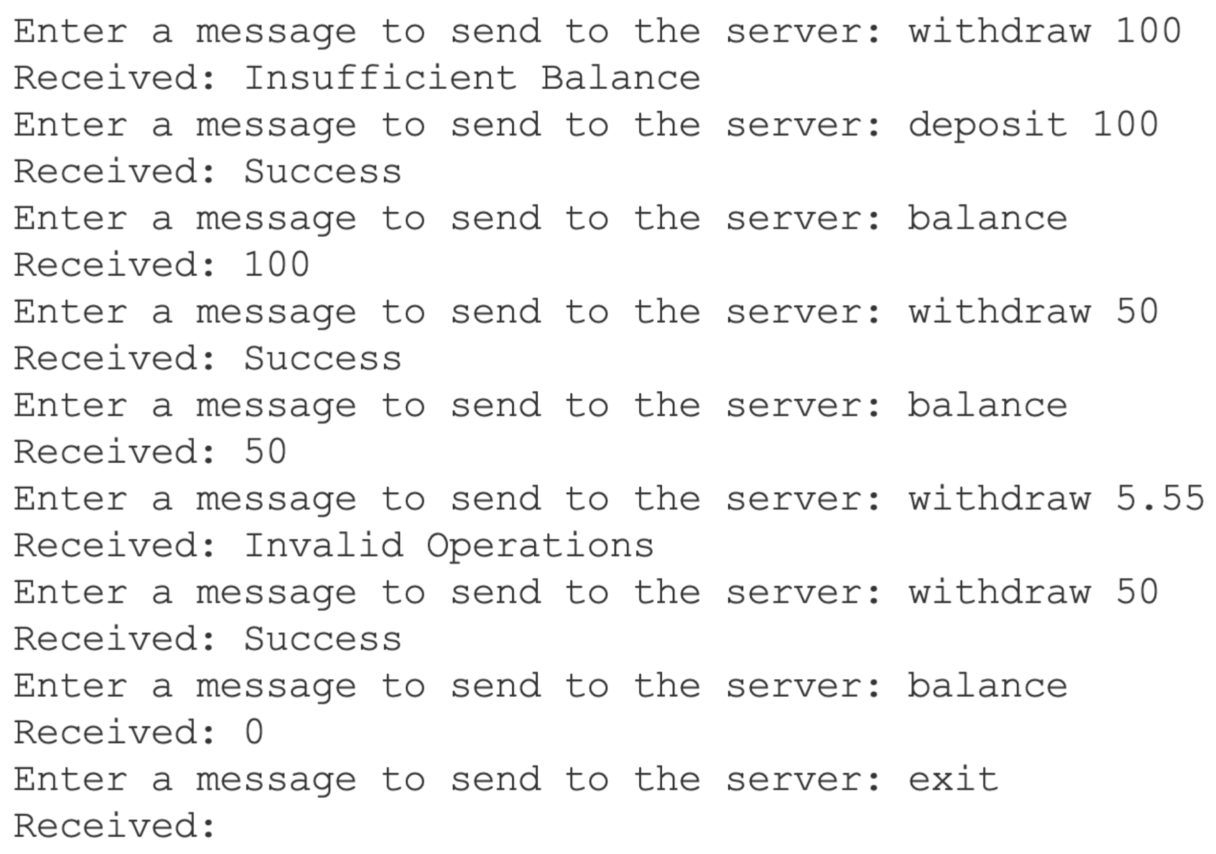
```
(base) keranwang@Kerans-MacBook-Pro v1.0 % python serve
r.py
Server initializing ...
Server has started and is listening on port 12345...
```

Figure 11. Server Initialization



```
● (base) keranwang@Kerans-MacBook-Pro v1.0 % python clie
nt.py
Client initializing ...
Enter 'create_account <username> <password>' to create
 an account
Enter 'login <username> <password>' to log in to an ac
count
create_account keran Qwer1234
The password must be at least 8 characters long.
The password must contain at least one uppercase lette
r, one lowercase letter, one digit, and one special ch
aracter.
Enter 'create_account <username> <password>' to create
 an account
Enter 'login <username> <password>' to log in to an ac
count
create_account keran Qwer1234^
Account created successfully.
Enter 'create_account <username> <password>' to create
 an account
Enter 'login <username> <password>' to log in to an ac
count
login keran Qwer1234^
Login successful.
Operations:      withdraw [$]
                 deposit [$]
                 balance
                 exit
```

Figure 12. Client (ATM) Account Creation and Login



```
Enter a message to send to the server: withdraw 100
Received: Insufficient Balance
Enter a message to send to the server: deposit 100
Received: Success
Enter a message to send to the server: balance
Received: 100
Enter a message to send to the server: withdraw 50
Received: Success
Enter a message to send to the server: balance
Received: 50
Enter a message to send to the server: withdraw 5.55
Received: Invalid Operations
Enter a message to send to the server: withdraw 50
Received: Success
Enter a message to send to the server: balance
Received: 0
Enter a message to send to the server: exit
Received:
```

Figure 13. Client (ATM) Operations