**Assignment 2**

**Contents**

[Program overview 1](#_Toc2620578)

[Screen captures for different Use Case scenarios 2](#_Toc2620579)

[UI1: Receiving funds from the bank 2](#_Toc2620580)

[UI2: Synchronizing two wallets 3](#_Toc2620581)

[UI3: Sending Funds 4](#_Toc2620582)

[UI4: Receiving Funds 5](#_Toc2620583)

[Discussion 5](#_Toc2620584)

[Vulnerability #1 5](#_Toc2620585)

[Vulnerability #2 5](#_Toc2620586)

# Program overview

**Bank\_1.py**

Considered to be the main class of this application. The Bank handles two wallets at a time and stores the wallets into its wallet array. The bank prompts the user to obtain Wallet A and Wallet B and allows the user to perform the different Use Case scenario options.

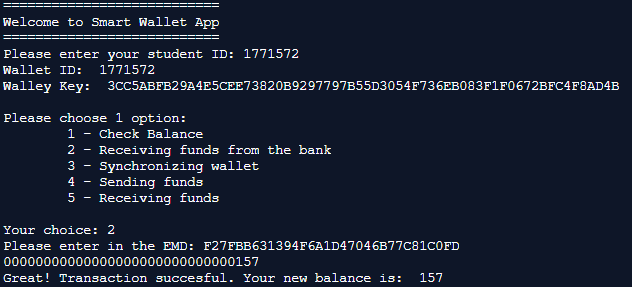
**Wallet.py**

Class that defines what a wallet is. The wallet contains its ID, balance, wallet key, and a table with synchronized wallets. A wallet is instantiated and consumed by the Bank\_1.py class.

Our EMD: F27FBB631394F6A1D47046B77C81C0FD

# Screen captures for different Use Case scenarios

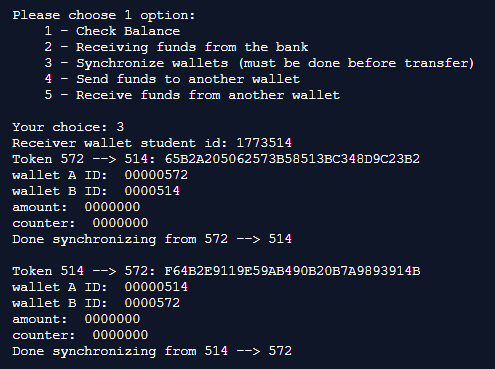
## UI1: Receiving funds from the bank



**Program flow description**

Upon start, the user is requested for their student ID. From this input, a wallet object (wallet\_a) is created, with its key, id, and balance attributes being initialized from the input provided and encrypted using SHA\_256.

The user (wallet\_a) is then displayed an option menu where they can receive funds from the bank. The program prompts the user for an Electronic Money Draft (EMD). This is then decrypted using the wallet’s secret key and will update the wallet’s balance accordingly.

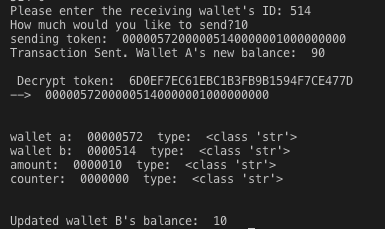
UI2: Synchronizing two wallets 

**Program flow description**

After wallet\_a has been initialized and funds have been added, the user will be able to synchronize with another wallet (a step that is needed before sending funds to another wallet).

The program prompts the user for the receiver’s wallet ID. With the knowledge of the sender (wallet\_a)’s wallet ID, the receiver (wallet\_b)’s wallet ID, and initial values of amount= 00000000, counter= 00000000, the token to be encrypted is generated and encrypted. The wallet\_a ID and counter is then stored into wallet\_b’s table. This is applied again with wallet\_b and wallet\_a reversing the sender/receiver roles.

## UI3: Sending Funds





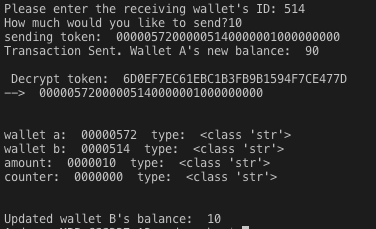
**Program flow description**

After two wallets have been synchronized, wallet\_a will be able to use this interface to transfer funds to wallet\_b. The user is prompted for the 3-digit Wallet ID of the receiving wallet (wallet\_b) along with the amount they would like to send. The program first verifies that wallet\_a holds sufficient funds for the transaction to occur. If this is true, the token to be encrypted is generated with the sender’s ID (wallet\_a), receiver’s ID (wallet\_b), amount to be sent, and the counter of B that is stored in wallet\_a’s table.

The [receiving\_funds()](#_UI4:_Receiving_Funds) method is then called.

The counter of B is then incremented by 1 in wallet\_a’s table.

## UI4: Receiving Funds





**Program flow description**

The token generated by the [sending\_funds()](#UI3: Sending Funds) method is then decrypted to obtain the ID of the sending wallet (wallet\_a), receiving wallet (wallet\_b), the amount being received, and the counter of CB from wallet\_a’s table. The program first checks to make sure that wallet\_b’s ID is indeed the wallet in the receiving id sent over by the token. The program also checks to make sure that the counter matches that in the record associated with wallet\_a’s ID. The balance for wallet\_b is then updated accordingly, and the CA is incremented by 1 in wallet\_b’s table.

# Discussion

Indicate 2 possible vulnerabilities in the current design and propose modifications to close such risk.

## Vulnerability #1- Use of ECB

ECB Mode encryption is not semantically secure, meaning that by observing ECB-encrypted ciphertext can leak information about the plaintext. The encryption of the same block of plaintext always yields in the same block of ciphertext, allowing the attacker to detect whether two ECB-encrypted messages are identical, if they share the same prefix, or other common substrings.

**Propose modification:**

Use authenticated encryption mode instead (such as OCB) which simultaneously assures the confidentiality and authenticity of data. Traditional semantically secure encryption modes such as CBC or CTR combined with a message authentication code (e.g. HMAC) could also be helpful.

## Vulnerability #2

Propose modification: