

CECS 478  
PHASE DESIGN  
“DUODOLO”  
RYAN VALDRIZ  
ROBERT SANDERSON

1. Assets
  - a. User Information
  - b. Content of messages
  - c. Server
  - d. Database
2. Stakeholders
  - a. Members of DuoDolo
  - b. Users of our messaging app
3. Adversary Models
  - a. User Info / DuoDolo & User Info/Users
    - i. Active Insider / Passive Insider
      1. Digital Resources
        - a. Low/high computational power
        - b. Low/high power supply
        - c. Might have access to database
        - d. Access to our code
      2. Protection Plan
        - a. Access Control
        - b. Need to implement accountability
        - c. Need to implement undeniability

- 1. logs/transcripts
- ii. Active Outsider
  - 1. Digital Resources
    - a. Low/High computational power
    - b. Low/High power supply
  - 2. Protection Plan
    - a. Limit number of attempts for login
    - b. Hash passwords
- iii. Passive Outsider
  - 1. Digital Resources
    - a. Low/High computational power
    - b. Low/High power supply
  - 2. Protection Plan
    - a. Hash passwords
    - b. Do not store information on server
- b. Message Content / DuoDolo & Message Content / User
  - i. Active Insider / Passive Insider
    - 1. Digital Resources
      - a. Low/high computational power
      - b. Low/high power supply
    - 2. Protection Plan
      - a. Encrypt messages
      - b. Encrypt keys
      - c. HMAC
  - ii. Active Outsider / Passive Outsider
    - 1. Digital Resources
      - a. Low/high computational power

- b. Low/high power supply

- 2. Protection Plan

- a. Encrypt messages

- b. Encrypt keys

- c. HMAC

- c. Server / DuoDolo

- i. Active Insider / Passive Insider

- 1. Digital Resources

- a. Low/high computational power

- b. Low/high power supply

- 2. Protection Plan

- a. AWS shield

- b. Accountability

- c. Undeniability

- i. Logs

- d. Access Control

- ii. Active Outsider / Passive Outsider

- 1. Digital Resources

- a. Low/high computational power

- b. Low/high power supply

- 2. Protection Plan

- a. AWS shield

- d. Server / User

- i. Active Outsider / Passive Outsider

- 1. Digital Resources

- a. Low/high computational power

- b. Low/high power supply

## 2. Protection Plan

### a. AWS shield

## e. Database / DuoDolo

### i. Active Insider / Passive Insider & Active Outsider / Passive Outsider

#### 1. Digital Resources

##### a. Low/high computational power

##### b. Low/high power supply

#### 2. Protection Plan

##### a. Accountability

##### b. Undeniability

##### c. Access Control

## f. Database / User

### i. Active Outsider / Passive Outsider

#### 1. Digital Resources

##### a. Low/high computational power

##### b. Low/high power supply

#### 2. Protection Plan

##### a. Encryption

##### b. Hash

## 4. Attack Surfaces

### a. Login Credentials

#### i. Brute force attacks

#### ii. Social Engineering

#### iii. Users

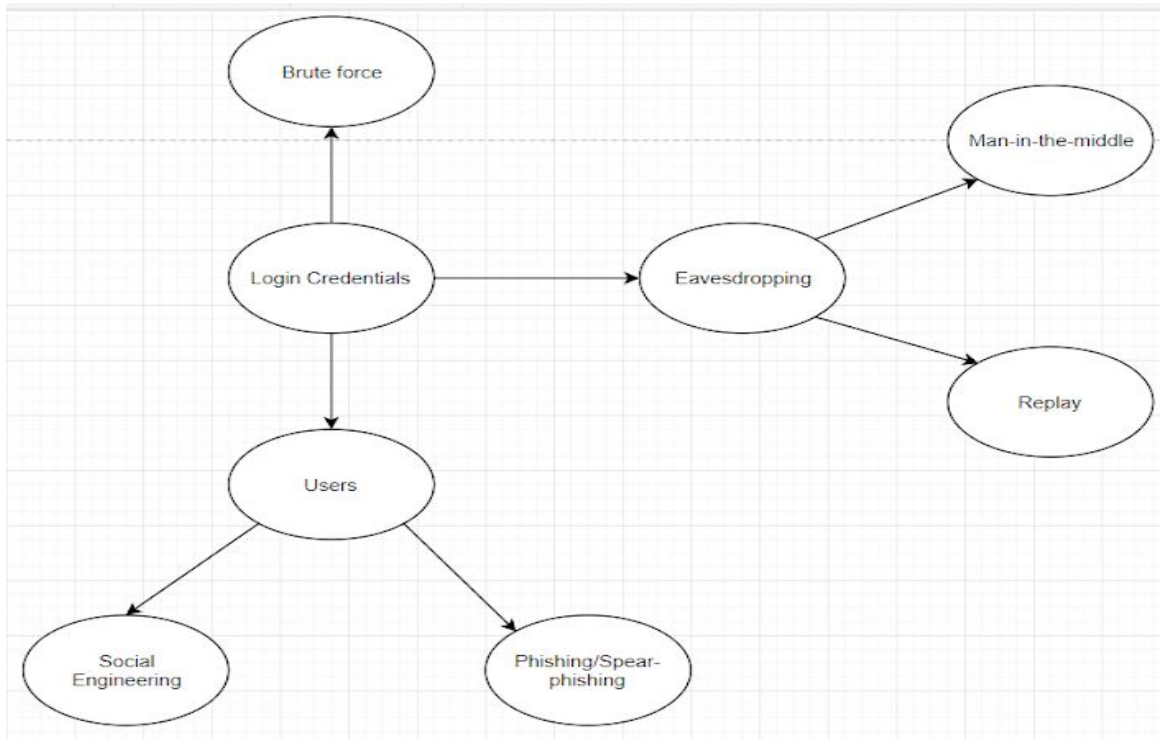
##### 1. Social Engineering

##### 2. Phishing / Spear phishing

#### iv. Eavesdropping

1. Man in the middle attack

2. Replay

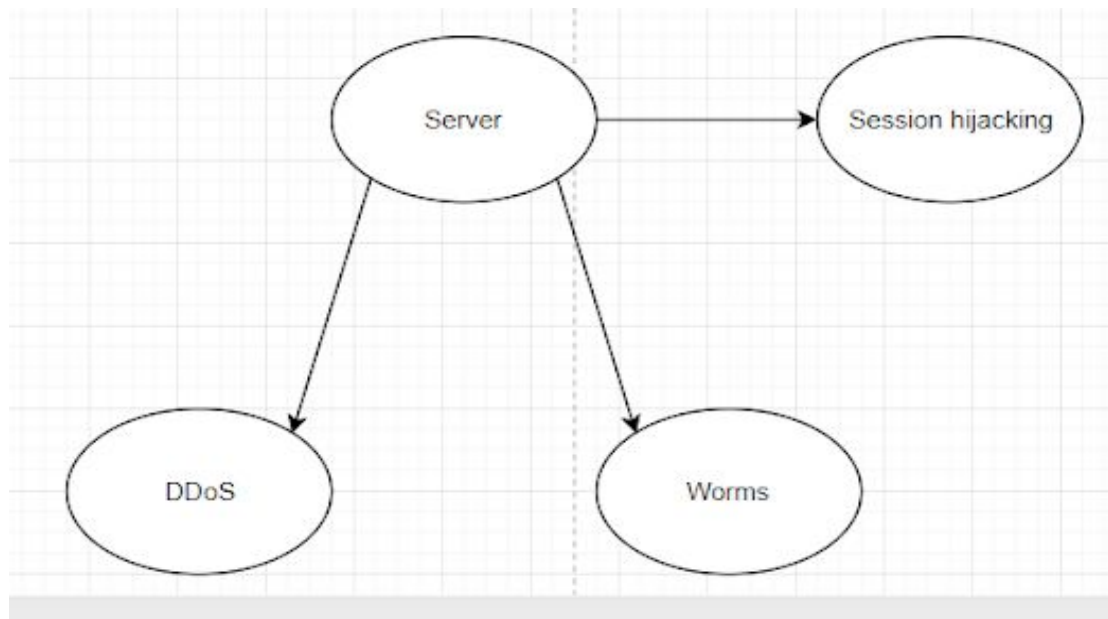


#### b. Server

i. Worms

ii. DDoS

iii. Session hijacking



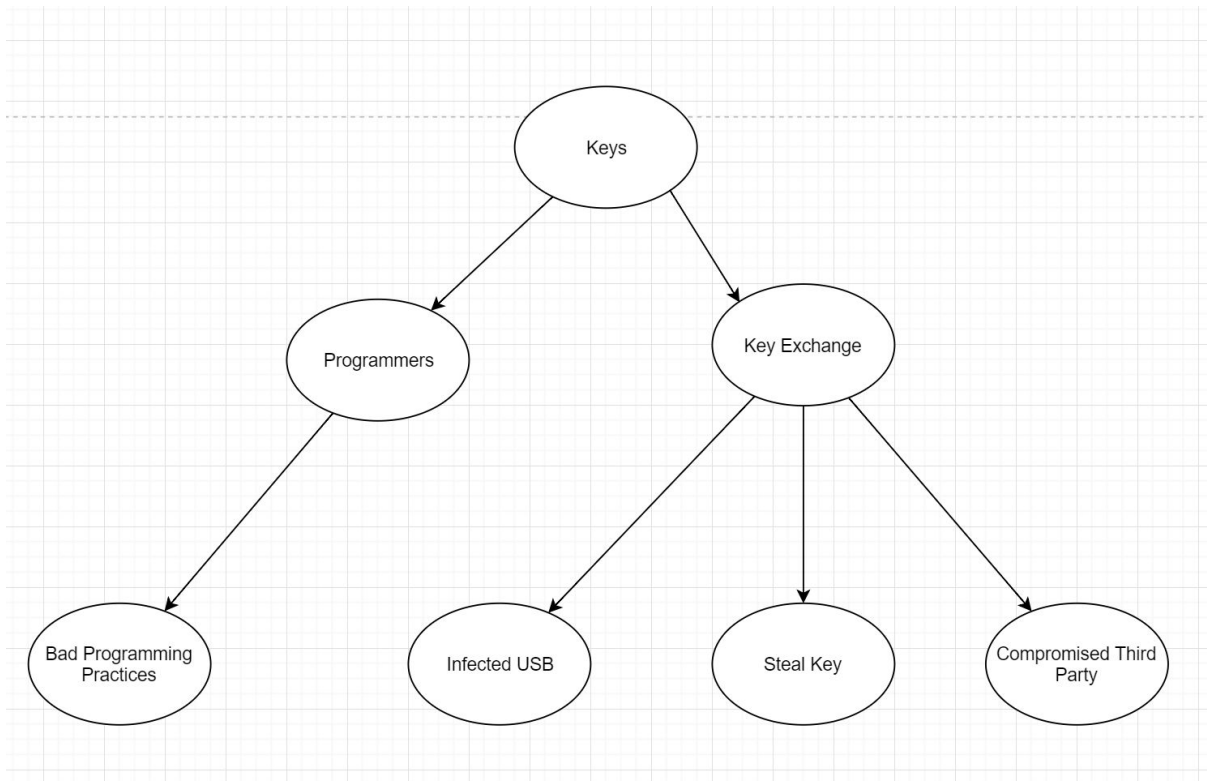
c. Key

i. Key Exchange

1. Steal Key
2. Infected USB
3. Compromised Third Party

ii. Programmers

1. Bad programming practices



## 5. Solutions

### a. Authentication Methods

- i. JWT tokens verify the identity of the user and gives access to send/receive messages.
- ii. Let's Encrypt
  1. Provides HTTPS domain authentication

### b. Message Integrity Check

- i. Integrity tag is generated by running HMAC on the ciphertext.
- ii. This tag is output to a JSON file with an encrypted concatenation of the HMAC and AES keys.
- iii. On decryption, the encrypted keys are decrypted.
- iv. HMAC is executed with the recovered HMAC key to re-generate the HMAC tag.

- v. This regenerated tag is compared with the recovered tag from the JSON.
- vi. If they match, then integrity is achieved.
- c. Preservation of confidentiality
  - i. Integrity checks
  - ii. Authentication
  - iii. Hashed passwords
  - iv. Encrypted messages
    - 1. A 256-bit AES key is used to encrypt the users message, which is followed up by an HMAC with a 256-bit HMAC key.
    - 2. The two keys, AES and HMAC, are then encrypted with an RSA public key provided by the user.

## 6. Analysis

- a. Pros
  - i. Provided PGP for messages.
  - ii. Only those communicating can read the messages as long as keys aren't compromised
    - 1. Server can't read messages
  - iii. RSA Keys are never passed through the server
  - iv. Deletes messages after reading them
    - 1. Adversaries cannot read messages that have already been read
  - v. **Professor Aliasgari designed most of the project.**
- b. Cons
  - i. Inconvenient way of exchanging keys
    - 1. In person

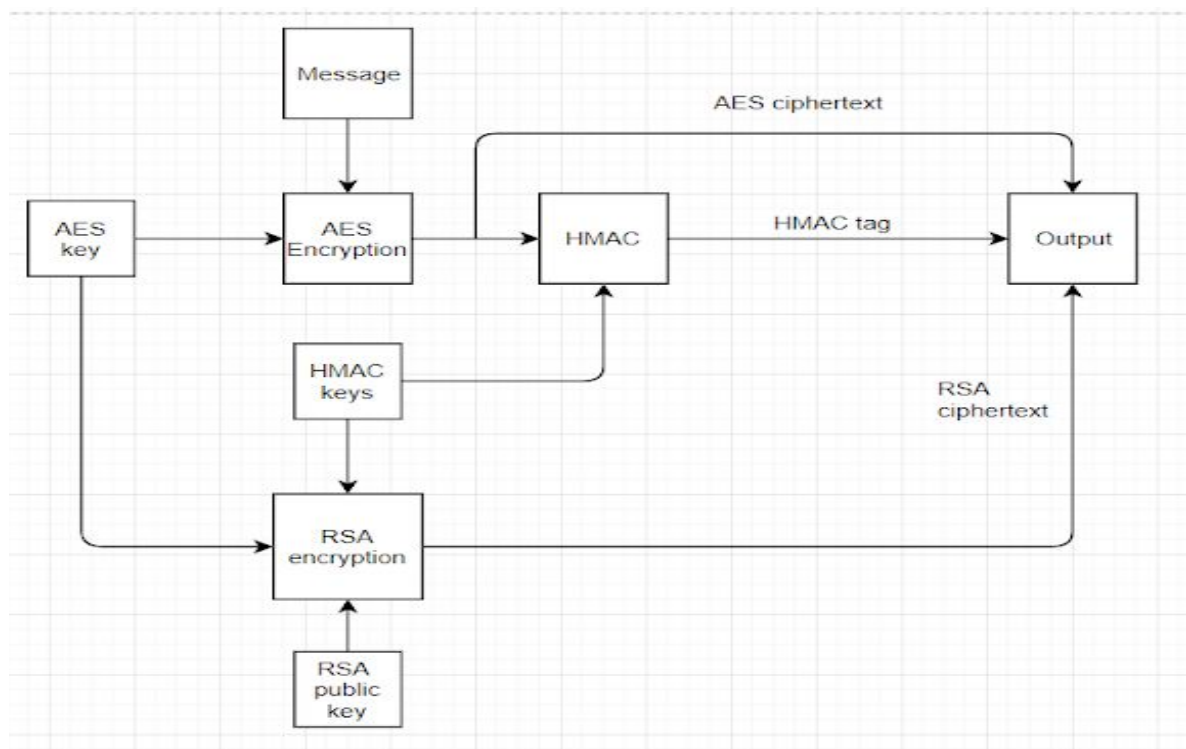


## ii. Not true end to end encryption

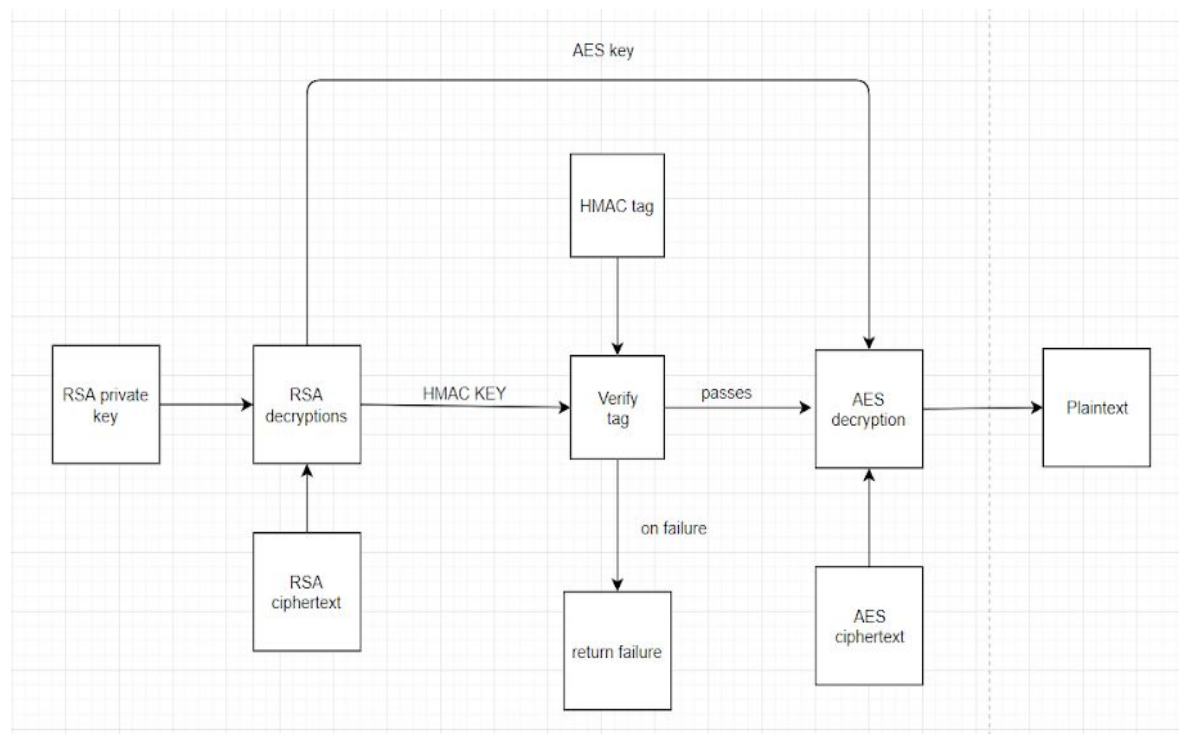
1. No forward secrecy
  2. Needs better method of key exchange
- iii. Requires user to provide path for public/private keys
- iv. Didn't implement a maximum number of password attempts to prevent brute forcing.
- v. Didn't use new keys every session (elliptic curve)
- vi. Unable to get the code fully functional on the server.
1. Worked on a localhost but not on the server.
- vii. No GUI
- viii. Deletes messages after reading them
1. Users might want to keep previous messages

## 7. System Diagrams

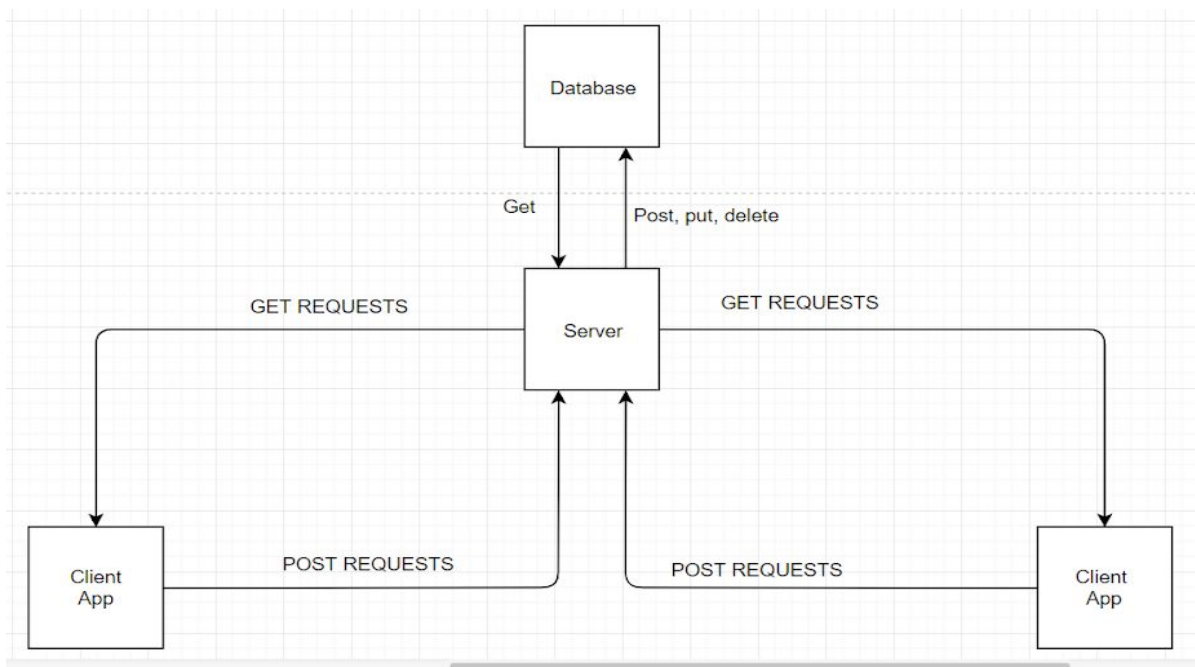
### a. Encrypting Messages



## b. Decrypting messages



## c. Overview



## 8. Future implementation

- a. Keep reusing jwt till it expires
  - i. Then require another login

- b. Time stamp
- c. GUI
  - i. Electron rather than a python terminal
- d. App key
  - i. Server only listens to the app we developed
- e. Ephemeral Keys
  - i. Key derivation functions
  - ii. Double-Ratchet algorithm
- f. Max number of password attempts to help deter brute forcing
- g. Auto generate key pair if not given by the user.
- h. Implement a way to not require the user to provide the path to a key file