COMP6841 - Extended Cyber Security and Security Engineering

William Gaston - z5591798

Short Report

avoDB

Results

My project, avoDB, delivers a command line interface (CLI) based DBaaS program which combines end-to-end encrypted data storage and retrieval functionality with a secure messaging system created on top of a robust authentication mechanism. I aimed my project to accommodate users who want full control over the encryption and decryption of their data, offering client-side encryption which does not rely on the server for any functionality other than accessing ciphertext within the database - ensuring the 5 pillars of information assurance are met.

The key features of my project include:

- End-to-end encrypted storage utilising symmetric and asymmetric algorithms
- Client-side encrypted messaging
- A robust key management system for generating and storing encryption keys
- A zero-knowledge backend server with minimal metadata exposure
- Secure authentication
- A modular repository structure posed for extension
- Message signing and cryptographic authentication / integrity checks

What I did

Client

The client holds the key modules used for user interfacing, cryptography, and session management. This directory holds the core implementation of the system's security practices - holding the logic of the encryption system. Outside of this, the larger program structure has no knowledge of how the encryption and hashing takes place and simply interfaces with the functions defined within.

Routes/Core

The routes/core module holds the primary logic of my application, in particular it provides the mechanism for which my client-side functions interact with the backend and server. It handles the enforcement of the client trust boundary - utilising the client-side routes to encrypt all plaintext prior to interaction with the server. Furthermore, the routes handle verification of database, table, and row ownership and authentication of messages and data.

Backend

The backend of avoDB is split into two main sections: the database server, the python query execution. My database is a custom PostgreSQL server running in a docker container - with inbuilt persistence via a data volume. To interact with this server, my backend python files utilise psycopg2 parameterised queries with a pool of connection objects which allow for concurrent operations.

Encryption Scheme

avoDB utilises a hybrid encryption system which combines the properties of both symmetric and asymmetric encryption techniques to strike a balance between security performance and efficiency. Through my scheme and overall design I ensure that the 5 pillars of information assurance are met.

- Symmetric Key Encryption - AES-GCM
- Asymmetric Key Encryption - RSA with OAEP
- Hashing - argon2
- OS keyring
- Key Generation

How I was Challenged - More challenges

Key Management Complexity

Issue: The complexity of my layered, hybrid key and encryption approach made it difficult to store, retrieve and utilise said keys without database knowledge.
Solution: I utilised local OS keyring modules for session management to reduce database overhead, and utilised encryption with the password as the sole point of failure.

Testing Difficulty

Issue: It is exceedingly difficult to robustly test cryptography using deterministic output, especially given the tight timeframe of the project.
Solution: I conducted extensive manual testing throughout development to ensure the different components function as expected.

Performance Reductions

Issue: Initially, it was difficult to use solely RSA asymmetric encryption as it would greatly reduce performance at higher scale.
Solution: I introduced a hybrid encryption scheme using RSA to encrypt symmetric keys.

Complex Encryption Library

Issue: Navigating the python provided cryptography libraries is exceedingly confusing and difficult without prior knowledge and exposure.
Solution: I read a multitude of documentation and completed trial and error until I found a working solution.

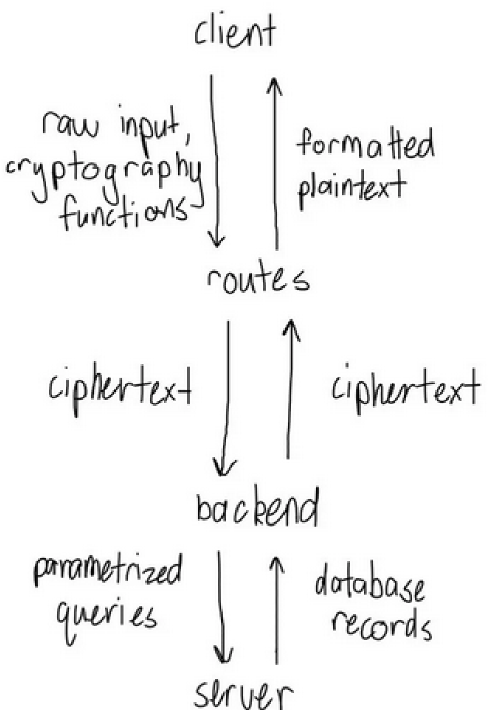
This experience was very valuable in allowing me to apply my theoretical knowledge and participate in self-guided learning in the security field. I found it challenging, but highly rewarding and greatly improved my security thinking and problem solving abilities. If I were to do it again I would attempt to implement my own encryption algorithm as a learning opportunity.

Appendix

CLI Overview

```
                  ____  ____  
  __ ___   _____ |  _ \| __ ) 
 / _` \ \ / / _ \| | | |  _ \ 
| (_| |\ V / (_) | |_| | |_) |
 \__,_| \_/ \___/|____/|____/ 

---------------------------------------------------------
| avoDB: an end-to-end encrypted database as a service. |
---------------------------------------------------------

user commands:
register       --username <*username*> --password <*password*>
login          --username <*username*> --password <*password*>
userList
logout

message commands:
initiateConvo  --userId <*userId*>
viewConvos
sendMsg        --message <*message*>
viewMsgs

database commands:
dbCreate       --name <*name*>
dbList
dbDelete       --dbId <*dbId*>

table commands:
tbCreate       --dbId <*dbId*> --name <*name*> --schema <*schema*>
tbList         --dbId <*dbId*>
tbSchema       --dbId <*dbId*> --tbId <*tbId*>
tbDelete       --dbId <*dbId*> --tbId <*tbId*>

data/row commands
insert       --dbId <*dbId*> --tbId <*tbId*> --data <*data*>
select       --dbId <*dbId*> --tbId <*tbId*>
rwList       --dbId <*dbId*> --tbId <*tbId*>
rwDelete     --dbId <*dbId*> --tbId <*tbId*> --rwId <*rwId*>
```

Workflow



This section provides screenshots of the main interface and corresponding output of my program along with code snippets of the different types of database operations in my application, including:

- Creation
- Selection
- Deletion

I also provided some evidence of my encryption and session management scheme.

The primary entry point to my program is through the main.py file which prompts the user the following choices:

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py
Usage: main.py [OPTIONS] COMMAND [ARGS]...

  avoDB: an end-to-end encrypted database as a service.

Options:
  --help  Show this message and exit.

Commands:
  auth
  db
  msg
  rw
  tb
```

Each set of cli options and commands is created like so:

```
import click

from api.routes.auth import *

@click.group()
def auth():
  pass

@click.command(help="--username <*username*> --password <*password*>")
@click.option('--username', prompt=True, help="your username")
@click.option('--password', prompt=True, hide_input=True, help="your password")
def login(username, password):
  loginFunc(username, password)

@click.command(help="--username <*username*> --password <*password*>")
@click.option('--username', prompt=True, help="your username")
@click.option('--password', prompt=True, hide_input=True, help="your password")
def register(username, password):
  registerFunc(username, password)

@click.command(help="")
def userList():
  userListFunc()

@click.command(help="")
def logout():
  logoutFunc()

auth.add_command(login)
auth.add_command(register)
auth.add_command(logout)
auth.add_command(userList)
```

1. User Authentication

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py auth
Usage: main.py auth [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  login     --username <*username*> --password <*password*>
  logout
  register  --username <*username*> --password <*password*>
  userlist
```

1.1. Registration

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py auth register
Username: test
Password:
Registration Successful
successfully added credentials to keyring
```

```python
def registerFunc(username, password):
  # 1. query db to check if username exists
  # 2 create uuid
  # 3. create key pairs
  # 4. serialize key pairs
  # 5. create kdf salt
  # 6. create pk iv
  # 7. hash password
  # 8. encrypt private key
  # 9. save keys for session

  checkIfLoggedIn()

  userNameExists = checkUsernameExits(username)
  if userNameExists:
    print('Username already exists, please choose another')
    return False

  userId = generateUserId()
  privateKey, publicKey = generateKeyPair()
  privateSerialisedKey, publicSerialisedKey = serialiseKeyPair(publicKey, privateKey, password)
  salt = generateSalt()
  iv = generateIV()
  hashedPassword = hashPassword(password)
  encryptedPrivateKey = encryptPrivateKey(privateSerialisedKey, password, salt, iv)

  encodedEncryptedPrivateKey = base64.b64encode(encryptedPrivateKey).decode('utf-8')
  encodedPublicKey = base64.b64encode(publicSerialisedKey).decode('utf-8')

  successful = addUserToDB(userId, username, hashedPassword, salt, iv, encodedEncryptedPrivateKey, encodedPublicKey, datetime.datetime.now())
  if successful:
    print('Registration Successful')
    setCredentials(userId, privateSerialisedKey, publicSerialisedKey, password, iv)
  else:
    print('Failed To Register')
```

1.2. Login

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py auth login
Username: test
Password:
Already logged in
```

```python
def loginFunc(username, password):
  # 1. query db to check if username exists
  # 2. return the hashed password from the db
  # 3. verify hashed password
  # 4. save keys for session

  checkIfLoggedIn()

  userNameExists = checkUsernameExits(username)
  if not userNameExists:
    print('Username does not exist')
    return

  userId = getUserId(username)
  hashedPassword = getHashedPassword(userId)
  if not verifyPassword(hashedPassword, password):
    print('Incorrect password given')
    return

  # get private key
  encryptedPrivateKey, publicKey, salt, iv = getUserData(userId)
  encryptedPrivateKey = base64.b64decode(encryptedPrivateKey)
  publicKey = base64.b64decode(publicKey)
  privateKey = decryptPrivateKey(encryptedPrivateKey, password, salt, iv)
  setCredentials(userId, privateKey, publicKey, password, iv)

  print('Successfully Logged in')
```

1.3. Userlist

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py auth userlist

Users:

username    userId
----------  ------------------------------------
will        64394da6-e804-4623-a8be-6265bca045fb
admin       42db0783-ff2d-43a9-8041-8f1921d992fa
test        20891b6f-092b-4482-b874-0b628f359e35
```

1.4. Logout

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py auth logout
Successfully logged out
```

2. Databases

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py db
Usage: main.py db [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  dbcreate  --name <*name*>
  dbdelete  --dbId <*dbId*>
  dblist
```

2.1. Creation

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ avodb db dbcreate
Name: myNewDb
Successfully added database
```

2.2. Deletion

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py db dbdelete
Dbid: 19a65cbb-d594-4fa7-8bd8-b0e23ac588b1
Successfully deleted db
```

2.3. List

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py db dblist
DB_id                                 dbName
------------------------------------  --------
19a65cbb-d594-4fa7-8bd8-b0e23ac588b1  mydb
```

3. Tables

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py tb
Usage: main.py tb [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  tbcreate  --dbId <*dbId*> --name <*name*> --schema <*schema*>
  tbdelete  --dbId <*dbId*> --tbId <*tbId*>
  tblist    --dbId <*dbId*>
  tbschema  --dbId <*dbId*> --tbId <*tbId*>
```

3.1. Creation

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py tb tbcreate
Dbid: 19a65cbb-d594-4fa7-8bd8-b0e23ac588b1
Name: mytable
Schema: name,message
Successfully added table
```

```python
def addTable(tableId, dbId, tbName, schema):
  qry = "insert into Tables(table_id, db_id, encrypted_table_name, encrypted_schema) values(%s, %s, %s, %s) returning db_id"
  cursor, connection = cursorCreation()

  try:
    cursor.execute(qry, [tableId, dbId, tbName, schema])
    connection.commit()
  except Exception as e:
    connection.rollback()
    print('Table Creation Failed: ', e)

  value = cursor.fetchone()
  cursorRemoval(cursor, connection)

  if value is None:
    print('failed to add table')
    sys.exit(1)

  print('Successfully added table')
  return True
```

3.2. Deletion

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py tb tbdelete
Dbid: 19a65cbb-d594-4fa7-8bd8-b0e23ac588b1
Tbid: 8597132c-c065-44fe-bd31-a40d290c9f8f
Successfully deleted table
```

3.3. List

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py tb tblist
Dbid: 19a65cbb-d594-4fa7-8bd8-b0e23ac588b1
tableId                                tableName    tableSchema
------------------------------------   -----------  ------------
8597132c-c065-44fe-bd31-a40d290c9f8f   mytable      name,message
```

3.4. Schema

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py tb tbschema
Dbid: 19a65cbb-d594-4fa7-8bd8-b0e23ac588b1
Tbid: 8597132c-c065-44fe-bd31-a40d290c9f8f

Schema:
 name,message
```

4. Rows

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py rw
Usage: main.py rw [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  insert    --dbId <*dbId*> --tbId <*tbId*> --data <*data*>
  rwdelete  --dbId <*dbId*> --tbId <*tbId*> --rwId <*rwId*>
  rwlist    --dbId <*dbId*> --tbId <*tbId*>
  select    --dbId <*dbId*> --tbId <*tbId*>
```

4.1. Creation

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py rw insert
Dbid: 19a65cbb-d594-4fa7-8bd8-b0e23ac588b1
Tbid: 8597132c-c065-44fe-bd31-a40d290c9f8f
Data: will,mymessage
Successfully added row
```

```python
def rwInsertRoute(dbId, tbId, data):
    # 1. check tb belongs to user
    # 2. validate data to schema
    # 3. generate iv
    # 4. generate rowId
    # 5. encrypt data
    # 6. check follows schema
    # 7. insert
    if checkDBBelongsToUser(getUserID(), dbId) == 0:
        print('Database does not belong to you. Please choose another')
        sys.exit(1)
    if checkTBBelongsToUser(getUserID(), tbId) == 0:
        print('Table does not belong to you/this database or does not exist. Please choose another')
        sys.exit(1)

    iv = generateIV()
    rowId = generateUserId()

    encryptedMasterKey = base64.b64decode(getMasterKey(dbId))
    masterKey = decryptWithPrivateKey(getPrivateKey(), encryptedMasterKey, getPassword())

    encryptedData = encryptMessage(data, iv, masterKey)

    insertRow(rowId, tbId, encryptedData, iv)
```

```python
def insertRow(rowId, tbId, encryptedData, iv):
    qry = "insert into Rows(row_id, table_id, encrypted_data, iv) values(%s, %s, %s, %s) returning row_id"
    cursor, connection = cursorCreation()

    try:
        cursor.execute(qry, [rowId, tbId, encryptedData, iv])
        connection.commit()
    except Exception as e:
        connection.rollback()
        print('Row Insertion Failed: ', e)

    value = cursor.fetchone()
    cursorRemoval(cursor, connection)

    if value is None:
        print('failed to insert row')
        sys.exit(1)

    print('Successfully added row')
    return True
```

4.2. Deletion

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py rw rwdelete
Dbid: 19a65cbb-d594-4fa7-8bd8-b0e23ac588b1
Tbid: 8597132c-c065-44fe-bd31-a40d290c9f8f
Rwid: de79f91c-9a01-48c4-bbcb-dcd7e4c47411
Successfully deleted row
```

```
def deleteRow(tbId, rwId):
  qry = "delete from Rows where table_id = %s and row_id = %s"
  cursor, connection = cursorCreation()

  try:
    cursor.execute(qry, [tbId, rwId])
    connection.commit()
  except Exception as e:
    connection.rollback()
    print('row deletion failed:', e)

  cursorRemoval(cursor, connection)

  print('Successfully deleted row')
  return True
```

4.3. Select

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py rw select
Dbid: 19a65cbb-d594-4fa7-8bd8-b0e23ac588b1
Tbid: 8597132c-c065-44fe-bd31-a40d290c9f8f

mytable:

name    message
------  ---------
will    mymessage
blah1   blah2
```

4.4. List

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py rw rwlist
Dbid: 19a65cbb-d594-4fa7-8bd8-b0e23ac588b1
Tbid: 8597132c-c065-44fe-bd31-a40d290c9f8f

mytable:

rowId                                 name
------------------------------------  --------------
de79f91c-9a01-48c4-bbcb-dcd7e4c47411  will,mymessage
```

5. Messaging

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py msg
Usage: main.py msg [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  initiateconvo  --userId <*userId*>
  sendmsg        --message <*message*>
  viewconvos
  viewmsgs
```

5.1. Initiate Conversation

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py msg initiateconvo
Userid: 42db0783-ff2d-43a9-8041-8f1921d992fa
Successfully initiated conversation
```

5.2. Send message

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py msg sendmsg
Message: heyyyy!!!!
Message sent successfully
```

5.3. View messages

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py msg viewmsgs
admin                         Me
--------------------------    -------------------
                              heyyyy
hi
how are you?
                              i am good thank you
                              how are you
i'm good too
i think my project is done
                              heyyyy!!!!
```

```python
def viewMsgsRoute():
  senderId = getUserID()
  recipientId = getConvoUserID()

  otherUser = getUsername(recipientId)

  encryptedUserSentMessages = viewMsgs(senderId, recipientId, True)

  userSentMessages = []

  for message in encryptedUserSentMessages:
    encryptedSenderKey = base64.b64decode(message[3])
    messageKey = decryptWithPrivateKey(getPrivateKey(), encryptedSenderKey, getPassword())
    decryptedMessage = decryptMessage(message[0], message[2], messageKey).decode('utf-8')
    sentAt = message[1]

    userSentMessages.append([decryptedMessage, sentAt])

  encryptedUserReceivedMessages = viewMsgs(recipientId, senderId, False)

  userReceivedMessages = []

  for message in encryptedUserReceivedMessages:
    encryptedSenderKey = base64.b64decode(message[3])
    messageKey = decryptWithPrivateKey(getPrivateKey(), encryptedSenderKey, getPassword())
    decryptedMessage = decryptMessage(message[0], message[2], messageKey).decode('utf-8')
    sentAt = message[1]

    userReceivedMessages.append([decryptedMessage, sentAt])

  taggedMessagesCurr = [('curr', timestamp, message) for message, timestamp in userSentMessages]
  taggedMessagesOther = [('other', timestamp, message) for message, timestamp in userReceivedMessages]

  combinedMessages = taggedMessagesCurr + taggedMessagesOther
  combinedMessages.sort(key=lambda x: x[1])

  rows = []
  for sender, timestamp, message in combinedMessages:
    if sender == 'curr':
      rows.append(["", message])
    else:
      rows.append([message, ""])

  print(tabulate(rows, headers=[f'{otherUser}', 'Me']))
```

1. View conversations

```
gaston@Surface4:~/Uni/COMP6841/avoDB$ python3 main.py msg viewconvos
username    userId
----------  ------------------------------------
admin       42db0783-ff2d-43a9-8041-8f1921d992fa
```

```python
def viewConvos(currUserId):
  qry = "select distinct u.username, u.user_id " \
  "from Messages m " \
  "join UsersMeta u ON ( " \
  "(m.receiver_id = u.user_id AND m.sender_id = %s) OR "\
  "(m.sender_id = u.user_id AND m.receiver_id = %s));"
  cursor, connection = cursorCreation()

  try:
    cursor.execute(qry, [currUserId, currUserId])
    connection.commit()
  except Exception as e:
    connection.rollback()
    print('Registration Failed: ', e)

  value = cursor.fetchall()
  cursorRemoval(cursor, connection)

  if value is None:
    return False

  return value
```

6. Cryptography

6.1. Symmetric Encryption

```python
def encryptMessage(data, iv, masterKey, privateKey):
    gcmKey = AESGCM(masterKey)
    iv = base64.b64decode(iv)
    deserialisedPrivateKey = deserialisePrivateKey(privateKey, getPassword())
    encoded_data = data.encode('utf-8')
    signature = getSignature(deserialisedPrivateKey, encoded_data)

    signedData = json.dumps({
        "data": base64.b64encode(encoded_data).decode('utf-8'),
        "signature": base64.b64encode(signature).decode('utf-8')
    }).encode('utf-8')

    cipherText = gcmKey.encrypt(iv, signedData, associated_data=None)

    return base64.b64encode(cipherText).decode('utf-8')
```

```python
def decryptMessage(ciphertext, iv, masterKey, publicKey):
    gcmKey = AESGCM(masterKey)
    iv = base64.b64decode(iv)
    try:
        decodedText = gcmKey.decrypt(iv, base64.b64decode(ciphertext), associated_data=None)
    except InvalidTag:
        print('Tampering of ciphertext detected')
        sys.exit(1)

    deBundledData = json.loads(decodedText.decode('utf-8'))
    data = base64.b64decode(deBundledData["data"])
    signature = base64.b64decode(deBundledData["signature"])

    deserialisedPublicKey = deserialisePublicKey(publicKey)

    try:
        verifySignature(deserialisedPublicKey, signature, data)
    except InvalidSignature:
        print('Invalid signature in data')
        sys.exit(1)

    return data
```

6.2. Asymmetric Encryption

```python
def encryptWithPublicKey(item, publicKey):
  deserialisedPublicKey = deserialisePublicKey(publicKey)
  encryptedItem = deserialisedPublicKey.encrypt(
    item,
    asym_padding.OAEP(
      mgf=padding.MGF1(algorithm=hashes.SHA256()),
      algorithm=hashes.SHA256(),
      label=None
    )
  )

  return encryptedItem

def decryptWithPrivateKey(privateKey, encryptedItem, password):
  deserialisedPrivateKey = deserialisePrivateKey(privateKey, password)
  decryptedItem = deserialisedPrivateKey.decrypt(
    encryptedItem,
    asym_padding.OAEP(
      mgf=padding.MGF1(algorithm=hashes.SHA256()),
      algorithm=hashes.SHA256(),
      label=None
    )
  )

  return decryptedItem
```

6.3. Key Generation/KDF

```python
import base64
import os
from cryptography.hazmat.primitives.asymmetric import rsa
import uuid

def generateMasterKey():
  masterKeyBytes = os.urandom(16)
  return masterKeyBytes

def generateSalt():
  salt = os.urandom(16)
  return base64.b64encode(salt).decode('utf-8')

def generateIV():
  iv = os.urandom(16)
  return base64.b64encode(iv).decode('utf-8')

def generateUserId():
  userId = uuid.uuid4()
  return str(userId)

def generateKeyPair():
  privateKey = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
  )

  publicKey = privateKey.public_key()
  return privateKey, publicKey
```

```python
import base64
from argon2.low_level import hash_secret_raw, verify_secret, Type

def derivePasswordKey(password, salt):
  hashedPassword =hash_secret_raw(
    password.encode(),
    base64.b64decode(salt),
    time_cost=2,
    memory_cost=64 * 1024,
    parallelism=4,
    hash_len=32,
    type=Type.D
  )
  return hashedPassword
```

6.4. Hashing

```python
from argon2 import PasswordHasher

hasher = PasswordHasher()

def hashPassword(password):
  hashedPassword = hasher.hash(password)
  return hashedPassword

def verifyPassword(hashedPassword, password):
  try:
    hasher.verify(hashedPassword, password.encode())
    return True
  except Exception:
    return False
```

6.5. Session Management

```python
def setCredentials(userId, privateKey, publicKey, password, iv):
  keyring.set_password(servicePrK, username, privateKey.decode())
  keyring.set_password(servicePbK, username, publicKey.decode())
  keyring.set_password(serviceP, username, password)
  keyring.set_password(serviceU, username, userId)
  keyring.set_password(serviceIV, username, iv)
  print('successfully added credentials to keyring')

def initiateConvo(userId, publicKey):
  keyring.set_password(convoUserId, username, userId)
  keyring.set_password(convoPublicKey, username, publicKey)

def getPrivateKey():
  secret = keyring.get_password(servicePrK, username)
  if secret is None:
    print("not logged in, please log in first")
    sys.exit(1)

  return secret.encode()

def getPublicKey():
  secret = keyring.get_password(servicePbK, username)
  if secret is None:
    print("not logged in, please log in first")
    sys.exit(1)

  return secret.encode()

def getPassword():
  secret = keyring.get_password(serviceP, username)
  if secret is None:
    print("not logged in, please log in first")
    sys.exit(1)

  return secret.encode()
```

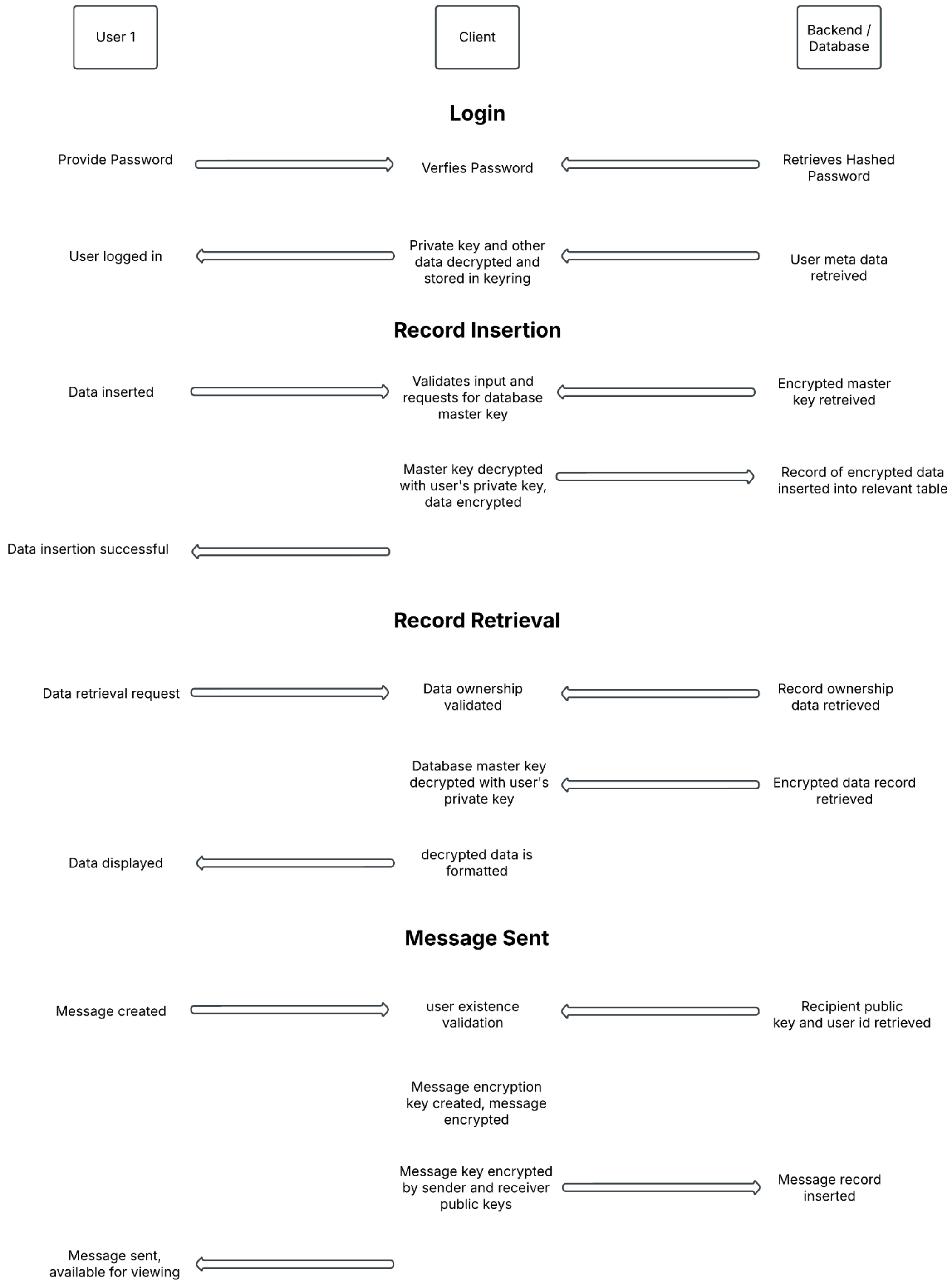
6.6. Authentication Signing

```python
def getSignature(privateKey, data):
    signature = privateKey.sign(
        data,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )

    return signature

def verifySignature(publicKey, signature, data):
    try:
        publicKey.verify(
            signature,
            data,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )
    except InvalidSignature:
        raise
```

Encryption Workflow

[plaintext] ===> sign ===> [plaintext , signature] ===> encrypt ===> [ciphertext]

[password] ===> kdf ===> [key symmetric key] ===> decrypt ===> [private key] ===> decrypt ===> [data symmetric key]



Schema

```sql
-- dump of database schema

create table if not exists UsersMeta (
  user_id               UUID primary key,
  username              text not null unique,
  kek_salt              text not null,
  pk_iv                 text not null,
  hashed_password       text not null,
  encrypted_private_key text not null,
  public_key            text not null,
  created_at            timestamp not null default current_timestamp
);

create table if not exists Databases (
  db_id                 UUID primary key,
  owner_id              UUID not null references UsersMeta(user_id) on delete cascade,
  iv                    text not null,
  encrypted_db_name     text not null,
  encrypted_master_key  text not null
);

create table if not exists Tables (
  table_id              UUID primary key,
  db_id                 UUID not null references Databases(db_id) on delete cascade,
  encrypted_table_name  text not null,
  encrypted_schema      text not null
);

create table if not exists Rows (
  row_id                UUID not null primary key,
  table_id              UUID not null references Tables(table_id) on delete cascade,
  encrypted_data        text not null,
  iv                    text not null,
  created_at            timestamp not null default current_timestamp
);

create table if not exists Messages (
  message_id              UUID not null primary key,
  encrypted_message       text not null,
  sender_id               UUID not null references UsersMeta(user_id),
  receiver_id             UUID not null references UsersMeta(user_id),
  sent_at                 timestamp not null default current_timestamp,
  encrypted_sender_key    text not null,
  encrypted_receiver_key  text not null,
  iv                      text not null
)
```

5 Pillars of Information Assurance Outline

1. **Confidentiality**
   All database records are encrypted client-side using AES-GCM (Advanced Encryption Algorithm with Galois/Counter Mode), ensuring end-to-end protection of database data. This guarantees the information input by the user has strong confidentiality both in transit and at rest on the server, preventing unauthorised access of the data even in the event the database is compromised.
   - Client-Side Encryption: data is encrypted on client's machine, thus plaintext is never exposed on the server
   - AES-GCM: The algorithm provides authenticated confidentiality with a high number of bits of work to guarantee security.
2. **Integrity**
   The integrity of the data used in my database is ensured through the authentication mechanism of AES-GCM which creates and verifies cryptographic authentication tags.
   - Tampering of the data and unauthorised modification (e.g. via changing bits) will cause the authentication tag check to fail.
3. **Authentication**
   Data access is tied to cryptographic key ownership and identity verification.

    - Password based encryption: data can only be correctly decrypted using the user's password (used to encrypt/decrypt the private key stored in the database), ensuring only the authorized user can access
    - GCM authentication tags: the algorithm provides authentication tags to ensure data originates from a legitimate source.
4. **Availability**
    - Containerized database: dockerized Postgresql enhances portability and resilience.
    - Data Persistence: volume mounting ensures database data is consistent across multiple instances.
5. **Non-repudiation**
   To establish non-repudiation, I comployed digital signatures which link the user's private key to their messages, verifiably linking actions to the users which perform them.
    - Private key signing: receivers of data can validate signatures using the user's public key, giving verifiable proof.

Other Challenges

Lack of Integrity/Authentication

Issue: My initial encryption scheme utilised AES-CBM which ensures confidentiality, but does not inherently support integrity or authentication. This makes my database and the data stored inside vulnerable to tampering with no method of ensuring the data is the same as that which was put into the database.
Solution: I conducted research and found that I had a couple of options.These included:

- message authentication code (MAC) which is a hash of the original message using a shared key and added to the stored data
- hashMAC (HMAC) where the shared key is used in the hashing process and not appended to the message. This is more desirable than a simple MAC as it is secure against reversal.
- AES Galois/Counter Mode (GCM) mode of operation builds on counter mode and implements a MAC using a GHASH function which uses Galois fields rather than primes. It is slightly weaker than a HMAC but significantly easier to implement and has the benefit of high performance with inexpensive resources.

I utilised the cryptography library primitives to implement the AES-GCM mode and algorithm.

Lack of Non-repudiation in Messaging

Issue: Using the hybrid encryption scheme of RSA and AES-GCM does not inherently provide signatures and guarantees of non-repudiation of messages within my messaging functionality.
Solution: I modified my encryption scheme to encrypt a JSON object containing the plaintext data and a signature created using the user's private key which could be verified with their

public key during decryption. If tampering is detected, an exception will be raised. I implemented this using the cryptography primitives library and the sign and verify functions of RSA objects.

- Conceptually, it was very challenging to create and design a suitable encryption scheme. The current complexity of my hybrid symmetric and asymmetric approach took significant research and self-guided study with little standardization.
- Understanding and applying cryptography theory was difficult, utilising dense libraries and trying to navigate the complexity of things like IVs, padding schemes, algorithms was very overwhelming.
- Debugging the encryption flow was fraught with challenges as it could not be easily reasoned with and the errors were often quite obscure.
- Approaching the task at a reasonable scale was difficult to navigate.