

**EASTERN MEDITERRANEAN UNIVERSITY**

**SPRING 24-25**

**CMSE456**

**TERM PROJECT:** WhatsApp Like Application (WALA)

**DEPARTMENT:** Software Engineering

**ADVISOR:** Alexander CHEFRANOV

**LAB-ASSISTANT:** Ali FARROKHNEJAD

**TEAM MEMBERS:** GR 9

Özhan DURMAZ 21330880

Kardelen TUMAY 20000110

Gözde ERTAŞ 21000054

Sinem DUMAN 21000052

**CITY/COUNTRY:** Turkish Republic of Northern Cyprus/ Gazimağusa

**OUTLINE**

PROBLEM DEFINITION………………………………………………………………………………1

WALA DETAILED DESCRIPTION …………………………………………………………...............1

SECURITY METHODS USED FOR WALA…………………………………………………………..1

DESCRIPTION OF WALA IMPLEMENTATION……………………………………………………..1

4.1 SYSTEM ARCHITECTURE OF 2 COMPUTERS………………………………………………...2

4.2 DATABASE STRUCTURE FOR WALA…………………………………………………………..2

4.3 SYSADMIN AND CLIENT………………………………………………………………………...2

4.4 DATA STRUCTURES FOR SECURE IMPLEMENTATION OF SYSADMIN AND CLIENT….2

4.5 ALGORITHMS USED FOR SECURE IMPLEMENTATION OF SYSADMIN AND CLIENT….3

4.6 DESCRIPTION OF CODES DEVELOPED……………………………………………………….3

5. TEST RESULTS……………………………………………………………………………………..4

5.1 Chat…………………………………………………………………………………………………4

5.2 SysAdmin Panel…………………………………………………………………………………….5

5.3 Adding User…………………………………………………………………………………………5

5.4 List the Users………………………………………………………………………………………..6

5.5 Delete User………………………………………………………………………………………….6

5.6 Show Previous Chats………………………………………………………………………………..7

5.7 Exit………………………………………………………………………………………………….7

7.CONCLUSION……………………………………………………………………………………….7

8.REFERENCES……………………………………………………………………………………….8

APPENDIX…………………………………………………………………………………………….9

**1-PROBLEM DEFINITION**

Develop an on-line WhatsApp-like application (WALA) on at least two separate computers providing chatting of the clients. WALA has two actors: Sysadmin, and Client. Sysadmin is responsible for monitoring WALA server. Clients communicate using DES-encrypted and digitally signed messages which are stored DES-encrypted. Session secret key is exchanged using RSA encryption. RSA keys are periodically updated.

**2-WALA DETAILED DESCRIPTION**

WALA (Wide Area Login and Administration) is a distributed system based on encryption developed to provide secure communication between the client and the server. The system protects data security with user authentication and message encryption processes, while also providing control authority to the system administrator. The client encrypts a randomly generated DES key with the RSA public key it receives from the server and sends it. This key ensures that messages between the two parties are encrypted and decrypted with the DES algorithm.

WALA stores user information and message history in the SQLite3 database; user passwords are protected with the MD5 hash algorithm. Messaging processes are carried out simultaneously using multiple threads. In addition, the system allows viewing user and message records with the administrator (SysAdmin) login. This terminal-based system offers a simple and functional solution for secure data transfer over the TCP/IP network

**3-SECURITY METHODS USED FOR WALA**

The WALA system was developed using the Python programming language. The system operates over TCP/IP protocol with socket programming (communication tool) to provide communication between the client and server. Threading module (synchronization tool) was used to ensure that messages are synchronized. User passwords were hashed with the MD5 algorithm (hash function) and stored in the SQLite database (database management tool). RSA (asymmetric encryption) and DES (symmetric encryption) algorithms were used for secure key sharing and for encrypting messages. At the beginning of the session in the system, the DES key encrypted with RSA is shared, then all messages are encrypted with DES. The system operates via the command line (terminal), and no web server is used. In addition, the system can be monitored and managed remotely thanks to the administrator (SysAdmin) mode (security tool and distributed system structure).

**4-DESCRIPTION OF WALA IMPLEMENTATION**

The codes were run via Windows PowerShell and VS Code , and the system performed user login, messaging, and management operations with terminal-based interaction. Multithreading was used to enable the client and server to send and receive messages simultaneously. SQLite was preferred as the database. WALA was implemented as a portable and independent Python application that could easily run on different computers.

1

**4.1 SYSTEM ARCHITECTURE OF 2 COMPUTERS**

The WALA system is built on a client-server architecture and is designed to work between at least two separate computers. One of the systems acts as a server while the other acts as a client. The server side accepts connections from the client, authenticates the user, and decrypts encrypted messages. The client side connects to the server, logs in the user, and sends encrypted messages. Communication between the two computers is carried out using socket programming with the TCP/IP protocol. Messages are encrypted with a randomly generated DES key by the client. This key is encrypted and sent with the server's RSA public key at the beginning of the session. The server decrypts the DES key with its own RSA private key and becomes able to decrypt messages. Sending and receiving messages can be done simultaneously using threading on both the client and server sides. Thanks to this structure, WALA offers a system architecture that supports secure, two-way, and real-time messaging between two different computers.

**4.2 DATABASE STRUCTURE FOR WALA**

Our WALA system uses the SQLite3 database to store user information and message history. SQLite is a file-based, lightweight database management system that does not require installation, so it is preferred in small and portable applications such as WALA. The database contains two main tables: users and messages. The users table consists of the username and password (in MD5 hash format) of each user and is used for authentication purposes in the system. The messages table is designed to record sent messages; this table contains fields such as the content of the message, the sender's username and time information. Thanks to this structure, the system can perform operations such as user login, authorization, message tracking and system control in a secure and orderly manner.

**4.3 SYSADMIN AND CLIENT**

There are two main user roles in the WALA system: Client and SysAdmin. Client is an ordinary user who logs into the system with his/her username and password. This user performs the sending and receiving of messages after connecting to the server. Client sends its messages by encrypting them securely with RSA after sharing the DES key and provides secure two-way communication by decrypting the incoming messages. On the other hand, SysAdmin is the authorized user who accesses the server side with a special login option when starting the system. SysAdmin can view all users in the database and the message history; thus, it can perform operations such as auditing the system, checking message records and security monitoring. In addition, SysAdmin can add new users when necessary or perform administrative tasks by observing the activities in the system. Thanks to this role separation, the system both provides secure communication between users and enables central management and control functions.

**4.4 DATA STRUCTURES FOR SECURE IMPLEMENTATION OF SYSADMIN AND CLIENT**

In the WALA system, various data structures and security measures have been used together in order for the Client and SysAdmin roles to work Decently. Firstly, user information is stored in the users table in the SQLite3 database. This table contains the username and password fields for each user; passwords are kept summarized using the MD5 hash algorithm for security. Thus, passwords are not stored as plaintext and are protected against data leaks.

2

When the client connects to the server, it enters the authentication process with its username and password. Verification is performed by comparing the login information with the hashed data in the database. In addition, the Client encrypts the DES key that it randomly generates with the RSA public key that it receives from the server at the beginning of the session. Thanks to this process, all messages between the client and the server are transmitted encrypted using the Dec algorithm.

The SysAdmin role, on the other hand, is activated on the server side with a special option when the system is started. The SysAdmin ID is defined with a fixed user name (for example, "admin") and is verified with the hashed password in the database during login. When SysAdmin logs into the system, admin\_tools.py it can access the data in the users and messages tables through it, so users and the entire message history can be managed.

In addition, encrypted keys and messages are processed as byte sequences throughout all sessions; no changes are made to the Octets sent without being decoded during data transfer. Thanks to this structure, both client and sysadmin roles can be operated in a secure, authorized and controllable

**4.5 ALGORITHMS USED FOR SECURE IMPLEMENTATION OF SYSADMIN AND CLIENT**

In the WALA system, both asymmetric and symmetric Decryption algorithms have been used together for the security of the client and the system administrator. After the client logs in to the system, it encrypts the RSA public key it receives from the server and the DES key it creates randomly and sends it to the server. Thanks to this process, the DES key reaches the server safely. Then, all messages between the client and the server are encrypted with Dec (Data Encryption Standard) algorithm. This algorithm is suitable for real-time messaging with its fast operation.

Users' passwords are summarized and saved to the database using the MD5 hash algorithm. Thus, passwords are not stored openly and data security is ensured. The system administrator (SysAdmin) also logs in by going through the same verification system, but after logging in, he can see the users and messages in the system. All these algorithms work together to ensure that both the client and the administrator process in an authoritative and secure manner.

**4.6 DESCRIPTION OF CODES DEVELOPED**

**server.py:** It represents the server side. It accepts client connections, receives the encrypted DES key with RSA, decrypts messages and saves them to the database.

**client.py:**Manages the client side. It connects to the server, logs in with the username and password, sends the DES key using RSA and initiates the encrypted message exchange.

**crypto\_utils.py:** It includes encryption and decryption operations. The RSA (for key exchange) and DES (for message encryption) algorithms are defined here.

**database.py**:SQLite manages database operations. Operations such as user verification, message recording, user listing are performed through this file.

**add\_user.py:** It is used to add new users to the database. Creates a registration with a username and MD5 hashed password.

**admin\_tools.py:** It is a module that only the system administrator can access. It provides the function of displaying all user information and message history.

**rsa\_auto\_rotate.py:**It is a utility script that allows RSA keys to be automatically recreated at December intervals. It keeps the key security up to date constantly.

3

**5. TEST RESULTS**

5.1 Chat

Here we tested chat, that is, the mutual transmission of messages between two PCs.We also made the DES key visible so that we can make sure that the encryption is working correctly.

Server:

metin, ekran görüntüsü, yazılım, işletim sistemi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

*Figure 1*

Client:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

*Figure 2*

4

5.2 SysAdmin Panel

SysAdmin must have features such as managing users and being able to see chat.It should be made sure that we display them properly in this panel

metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

*Figure 3*

5.3 Adding User

When we select this function, as admin, we should be asked for username and password for the user to be added. After assigning them, this person is registered in the system and is seen when we list it. But if there is a person already registered here, the system should give us a warning.

metin, ekran görüntüsü, yazılım içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

*Figure 4*

5

5.4 List the Users

When we list after adding a user, we see all the users currently registered in the system. This list is updated after each addition/deletion.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

*Figure 5*

5.5 Delete User

Admin also has the right to delete a user, and as mentioned in the previous test, after the deletion, that user will no longer appear in the database and in the list.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir. metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

*Figure 6 Figure 7*

6

5.6 Show Previous Chats

Admin can of course also see the past chats of the users.Here he is allowed to view up to 10 lines of chat history.The screenshot shows the last conversation

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

*Figure 8*

5.7 Exit

This section is used to log the user out of the system.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

*Figure 9*

7. CONCLUSION

In conclusion, the WALA system developed in this project provides a secure and auditable communication infrastructure by combining functions such as user authentication, encrypted messaging and centralized management. In the system developed in Python, RSA and DES algorithms are used together to provide both secure key sharing and fast message encryption. With SQLite database, user and message data are stored permanently, and simultaneous communication is established on the client and server side thanks to threading. In addition, the fact that the system administrator can see users and messages provides an important advantage in terms of security and manageability. WALA is a secure communication solution that is open to development and adaptable to different scenarios thanks to its modular structure. It is a project that has successfully passed the tests we have conducted and has reached the necessary competence.

7

8.REFERENCES

* Python Socket Programming

Python Software Foundation. (n.d.). *socket — Low-level networking interface*. Python Documentation.  
<https://docs.python.org/3/library/socket.html>

* Threading in Python

Python Software Foundation. (n.d.). *threading — Thread-based parallelism*.  
<https://docs.python.org/3/library/threading.html>

* PyCryptodome Kütüphanesi (RSA, DES)

PyCryptodome Documentation. (n.d.). *Cryptographic modules*.  
https://www.pycryptodome.org/src/

* SQLite Veritabanı Kullanımı

SQLite Documentation. (n.d.). *SQLite Home Page*.  
https://www.sqlite.org/index.html

* MD5 Hashing

Python Software Foundation. (n.d.). *hashlib — Secure hashes and message digests*.  
<https://docs.python.org/3/library/hashlib.html>

* RSA Key Exchange & Symmetric Encryption Concepts

Stallings, W. (2016). *Cryptography and Network Security: Principles and Practice* (7th Edition). Pearson Education.  
(Akademik kaynak – RSA ve DES kavramları için)

* Client-Server System Architecture

Tanenbaum, A. S., & Van Steen, M. (2007). *Distributed Systems: Principles and Paradigms*. Prentice-Hall.

**8**

**APPENDIX**

**Source Code Of server.py**

import socket

import threading

from crypto\_utils import decrypt, encrypt, rsa\_decrypt

from database import init\_db, check\_user, log\_message

from rsa\_auto\_rotate import auto\_generate\_rsa\_keys

auto\_generate\_rsa\_keys()

init\_db()

admin\_mode = input("🔧 Log in as SysAdmin ? (y/n): ").lower()

if admin\_mode == 'y':

    from admin\_tools import sysadmin\_panel

    sysadmin\_panel()

    exit()

username = input("🔐 Server username: ")

password = input("🔐 Password: ")

if not check\_user(username, password):

    print("❌ Invalid user info.")

    exit()

print("✅ Logged in.")

HOST = '0.0.0.0'

PORT = 12345

def receive\_messages(conn, key):

    while True:

        try:

            data = conn.recv(1024)

            if not data:

                break

            decrypted = decrypt(data, key)

            log\_message(sender="client", receiver=username, message=decrypted)

            print(f"🟢 Client: {decrypted}")

        except Exception as e:

            print("❌ receive\_messages error:", e)

            break

def send\_messages(conn, key):

    while True:

        try:

            message = input("🔵 Server (message): ")

            if message == "":

                break

            encrypted = encrypt(message, key)

            conn.sendall(encrypted)

            log\_message(sender=username, receiver="client", message=message)

        except Exception as e:

            print("❌ send\_messages error:", e)

            break

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

    s.bind((HOST, PORT))

    s.listen()

    print("🟢 Server ready , waiting for connection...")

    conn, addr = s.accept()

    with conn:

        print(f"🔗 Connected: {addr}")

        try:

            with open("rsa\_keys/public\_key.pem", "rb") as f:

                pub\_key\_data = f.read()

            print("📤 Public key file rewas read successfully . Size:", len(pub\_key\_data))

            conn.sendall(pub\_key\_data)

            print("📡 Public key sended to client.")

            encrypted\_key = b''

            while len(encrypted\_key) < 256:

                part = conn.recv(256 - len(encrypted\_key))

                if not part:

                    raise ValueError("missing key retrieval")

10

                encrypted\_key += part

            key = rsa\_decrypt(encrypted\_key, "rsa\_keys/private\_key.pem")

            print("✅ DES key solved.")

            print("🧪 Server DES Key:", key)

        except Exception as e:

            print("❌ key solving error:", e)

            conn.close()

            exit()

        threading.Thread(target=receive\_messages, args=(conn, key), daemon=True).start()

        send\_messages(conn, key)

**Source Code Of client.py**

import socket

import threading

from crypto\_utils import encrypt, decrypt, rsa\_encrypt, generate\_des\_key

from database import init\_db, log\_message

init\_db()

HOST = '192.168.0.111'

PORT = 12345

username = input("🔐 Username: ")

def receive\_messages(sock, key):

    while True:

        try:

            data = sock.recv(1024)

            if not data:

                break

            decrypted = decrypt(data, key)

            log\_message(sender="server", receiver=username, message=decrypted)

            print(f"🟢 Server: {decrypted}")

        except Exception as e:

            print("❌ receive\_messages error:", e)

            break

def send\_messages(sock, key):

    while True:

        try:

            message = input("🔵 Client (message): ")

            if message == "":

                break

            encrypted = encrypt(message, key)

            sock.sendall(encrypted)

            log\_message(sender=username, receiver="server", message=message)

        except Exception as e:

            print("❌ send\_messages error:", e)

            break

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

    s.connect((HOST, PORT))

    print("📡 Waiting public key from Server...")

    pub\_key = s.recv(1024)

    print("✅ Public key received. Size:", len(pub\_key))

    with open("temp\_public\_key.pem", "wb") as f:

        f.write(pub\_key)

    des\_key = generate\_des\_key()

    print("🧪 Client DES Key:", des\_key)

    encrypted\_key = rsa\_encrypt(des\_key, "temp\_public\_key.pem")

    s.sendall(encrypted\_key)

    print("🔐 DES key send encrypted.")

    threading.Thread(target=receive\_messages, args=(s, des\_key), daemon=True).start()

    send\_messages(s, des\_key)

**Source Code Of admin\_tools.py**

from database import check\_user, add\_user

import sqlite3

def sysadmin\_panel():

    print("🔐 SYSADMIN PANEL")

    username = input("Username: ")

    password = input("Password: ")

    if not check\_user(username, password):

        print("❌ Unauthorized access!")

        return

    while True:

        print("\n📊 [1] List Users")

        print("➕ [2] Add new user")

        print("🗑️  [3] Delete User")

        print("📜 [4] Show Messages")

        print("🚪 [0] Exit")

        choice = input("Select: ")

        if choice == "1":

            show\_users()

        elif choice == "2":

            new\_user = input("New user's name: ")

            new\_pass = input("password: ")

            add\_user(new\_user, new\_pass)

        elif choice == "3":

            delete\_user()

        elif choice == "4":

            show\_messages()

        elif choice == "0":

            break

        else:

            print("Invalid selection.")

def show\_users():

    conn = sqlite3.connect("wala.db")

    c = conn.cursor()

    c.execute("SELECT username FROM users")

    users = c.fetchall()

    print("\n👥 Users:")

    for user in users:

        print("- " + user[0])

    conn.close()

def delete\_user():

    uname = input("Name of the user to be deleted: ")

    conn = sqlite3.connect("wala.db")

    c = conn.cursor()

    c.execute("DELETE FROM users WHERE username=?", (uname,))

    conn.commit()

    print(f"🗑️  User '{uname}' deleted.")

    conn.close()

def show\_messages():

    conn = sqlite3.connect("wala.db")

    c = conn.cursor()

    c.execute("SELECT sender, receiver, message, timestamp FROM messages ORDER BY timestamp DESC LIMIT 10")

    logs = c.fetchall()

    print("\n📝 Last 10 Mesagge:")

    for row in logs:

        print(f"[{row[3]}] {row[0]} → {row[1]}: {row[2]}")

    conn.close()

**Source Code of database.py**

import sqlite3

import hashlib

def init\_db():

    conn = sqlite3.connect("wala.db")

    c = conn.cursor()

    c.execute("""

        CREATE TABLE IF NOT EXISTS users (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            username TEXT UNIQUE NOT NULL,

            password TEXT NOT NULL

        )

    """)

    c.execute("""

        CREATE TABLE IF NOT EXISTS messages (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            sender TEXT,

            receiver TEXT,

            message TEXT,

            timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP

        )

    """)

    conn.commit()

    conn.close()

def hash\_password(password):

    return hashlib.sha256(password.encode()).hexdigest()

def add\_user(username, password):

    conn = sqlite3.connect("wala.db")

    c = conn.cursor()

    try:

        c.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, hash\_password(password)))

        conn.commit()

    except sqlite3.IntegrityError:

        print("⚠️ This user already exist.")

    conn.close()

def check\_user(username, password):

    conn = sqlite3.connect("wala.db")

    c = conn.cursor()

    c.execute("SELECT \* FROM users WHERE username=? AND password=?", (username, hash\_password(password)))

    result = c.fetchone()

    conn.close()

    return result is not None

def log\_message(sender, receiver, message):

    conn = sqlite3.connect("wala.db")

    c = conn.cursor()

    c.execute("INSERT INTO messages (sender, receiver, message) VALUES (?, ?, ?)", (sender, receiver, message))

    conn.commit()

    conn.close()

**Source Code Of rsa\_auto\_rotate.py**

from Crypto.PublicKey import RSA

import os

def auto\_generate\_rsa\_keys(directory="rsa\_keys"):

    if not os.path.exists(directory):

        os.makedirs(directory)

    private\_path = os.path.join(directory, "private\_key.pem")

    public\_path = os.path.join(directory, "public\_key.pem")

    key = RSA.generate(2048)

    private\_key = key.export\_key()

    public\_key = key.publickey().export\_key()

    with open(private\_path, "wb") as priv\_file:

        priv\_file.write(private\_key)

    with open(public\_path, "wb") as pub\_file:

        pub\_file.write(public\_key)

    print("🔁 RSA keys renewed.")

if \_\_name\_\_ == "\_\_main\_\_":

    auto\_generate\_rsa\_keys()

**Source Code Of crypto\_utils.py**

from Crypto.Cipher import DES

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

from Crypto.Random import get\_random\_bytes

# DES anahtarı oluştur

def generate\_des\_key():

    return get\_random\_bytes(8)  # DES için 8 byte anahtar

# DES ile şifreleme

def des\_encrypt(message, key):

    cipher = DES.new(key, DES.MODE\_ECB)

    padded = message + ' ' \* (8 - len(message) % 8)

    return cipher.encrypt(padded.encode())

# DES ile çözme

def des\_decrypt(ciphertext, key):

    cipher = DES.new(key, DES.MODE\_ECB)

    decrypted = cipher.decrypt(ciphertext).decode().rstrip()

    return decrypted

# RSA ile şifreleme

def rsa\_encrypt(message, public\_key\_bytes):

    public\_key = RSA.import\_key(public\_key\_bytes)

    cipher = PKCS1\_OAEP.new(public\_key)

    return cipher.encrypt(message)

# RSA ile çözme

def rsa\_decrypt(ciphertext, private\_key\_bytes):

    private\_key = RSA.import\_key(private\_key\_bytes)

    cipher = PKCS1\_OAEP.new(private\_key)

    return cipher.decrypt(ciphertext)

from Crypto.Cipher import DES

import hashlib

def generate\_des\_key():

    # 8 baytlık (64 bit) anahtar üretir

    password = "sifreleme"

    key = hashlib.md5(password.encode()).digest()[:8]

    return key

def pad(text):

    while len(text) % 8 != 0:

        text += ' '

    return text

def encrypt(message, key):

    des = DES.new(key, DES.MODE\_ECB)

    padded\_text = pad(message)

    encrypted\_text = des.encrypt(padded\_text.encode())

    return encrypted\_text

def decrypt(ciphertext, key):

    des = DES.new(key, DES.MODE\_ECB)

    decrypted\_text = des.decrypt(ciphertext).decode().rstrip(' ')

    return decrypted\_text