MINI PROJECT

Hybrid Authentication System Documentation

Introduction

This documentation provides a comprehensive guide to creating a hybrid authentication system in Python, utilizing both symmetric and asymmetric encryption methods. The authentication system securely stores and verifies user passwords by encrypting them with a symmetric key, and it securely stores the symmetric key using asymmetric encryption. The project uses the `cryptography` library for encryption and decryption.

Encryption Methods Used

- **Symmetric Encryption**: Symmetric encryption is employed to encrypt and decrypt user passwords using a shared secret key. This method provides efficient encryption and decryption operations. The `Fernet` encryption scheme from the `cryptography` library is used for symmetric encryption.
- **Asymmetric Encryption**: Asymmetric encryption is utilized for securely transmitting and storing the symmetric key used for password encryption. This method involves a pair of public and private keys. The `RSA` encryption algorithm from the `cryptography` library is used for asymmetric encryption.

Prerequisites

Before proceeding, ensure you have the following installed on your system:

- Python (version 3.6 or higher)
- `cryptography` library

You can install the required library using pip:

```
pip install cryptography
```

Project Overview

The hybrid authentication system consists of the following components:

1. `authentication.py`: The main module that implements the authentication system.
2. MySQL Database: A relational database that stores user data (username, symmetrically encrypted passwords, and asymmetrically encrypted symmetric keys).

Implementation

Step 1: Set Up the MySQL Database

Before implementing the authentication system, create a MySQL database to store user data. Use tools like phpMyAdmin or MySQL Workbench to create the database and a table to hold user information.

For this documentation, assume the database is named `user_authentication_db`, and the table is named `users`. The `users` table has the following structure:

```
sqlCopy code
CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    password_encrypted TEXT NOT NULL
);
```

Step 2: Implement the Hybrid Authentication System

Create a Python module named `authentication.py`. This module provides the authentication functionalities, including user registration and verification, using a hybrid encryption approach.

Here's the implementation of `authentication.py`:

```
pythonCopy code
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.fernet import Fernet
import mysql.connector

class AuthenticationError(Exception):
    pass

class Authentication:
    def __init__(self, database_host, database_user, database_password, database_nam
e):
        self.connection = mysql.connector.connect(
```

```
            host=database_host,
            user=database_user,
            password=database_password,
            database=database_name,
        )
        self.create_table()

        # Generate asymmetric keys (RSA)
        self.private_key = rsa.generate_private_key(public_exponent=65537, key_size=20
48)
        self.public_key = self.private_key.public_key()

        # Generate symmetric key (Fernet)
        self.symmetric_key = Fernet.generate_key()
        self.symmetric_cipher_suite = Fernet(self.symmetric_key)

    def create_table(self):
        with self.connection.cursor() as cursor:
            cursor.execute(
                """
                CREATE TABLE IF NOT EXISTS users (
                    id INT AUTO_INCREMENT PRIMARY KEY,
                    username VARCHAR(255) UNIQUE NOT NULL,
                    password_encrypted TEXT NOT NULL,
                    symmetric_key_encrypted TEXT NOT NULL
                )
                """
            )
        self.connection.commit()

    def register_user(self, username, password):
        # Encrypt the symmetric key with the user's public key (asymmetric encryption)
        encrypted_symmetric_key = self.public_key.encrypt(
            self.symmetric_key,
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None,
            ),
        )

        # Encrypt the user password with the symmetric key (symmetric encryption)
        encrypted_password = self.symmetric_cipher_suite.encrypt(password.encode()).de
code()

        with self.connection.cursor() as cursor:
            cursor.execute(
                "INSERT INTO users (username, password_encrypted, symmetric_key_encryp
ted) VALUES (%s, %s, %s)",
                (username, encrypted_password, encrypted_symmetric_key),
            )
        self.connection.commit()

    def verify_user(self, username, password):
        with self.connection.cursor() as cursor:
            cursor.execute(
```

```
                "SELECT password_encrypted, symmetric_key_encrypted FROM users WHERE u
sername = %s", (username,)
            )
            row = cursor.fetchone()
            if row is None:
                raise AuthenticationError("User not found.")
            stored_encrypted_password, stored_encrypted_symmetric_key = row

            # Decrypt the symmetric key with the private key (asymmetric decryption)
            decrypted_symmetric_key = self.private_key.decrypt(
                stored_encrypted_symmetric_key,
                padding.OAEP(
                    mgf=padding.MGF1(algorithm=hashes.SHA256()),
                    algorithm=hashes.SHA256(),
                    label=None,
                ),
            )

            # Use the decrypted symmetric key to decrypt the user password (symmetric
decryption)
            decrypted_password = Fernet(decrypted_symmetric_key).decrypt(stored_encryp
ted_password.encode()).decode()

            if decrypted_password == password:
                print("Login successful.")
            else:
                raise AuthenticationError("Invalid password.")

    def close_connection(self):
        self.connection.close()
```

Usage

To use the hybrid authentication system in your project, follow these steps:

1. Import the `Authentication` class from `authentication.py`.

```
from authentication import Authentication
```

1. Create an instance of the `Authentication` class, passing your MySQL database credentials.

```
pythonCopy code
auth = Authentication(
    database_host="your_mysql_host",
    database_user="your_mysql_username",
    database_password="your_mysql_password",
```

```
    database_name="user_authentication_db",
)
```

Replace `"your_mysql_host"`, `"your_mysql_username"`, `"your_mysql_password"`, and `"user_authentication_db"` with your actual MySQL database credentials and the database name.

1. Register a new user by calling the `register_user` method.

```
pythonCopy code
auth.register_user("john_doe", "password123")
```

1. Verify a user's login credentials by calling the `verify_user` method.

```
pythonCopy code
try:
    auth.verify_user("john_doe", "password123")
    print("Login successful!")
except AuthenticationError as e:
    print(f"Authentication Error: {e}")
```

1. Close the database connection when you're done with the authentication system.

```
pythonCopy code
auth.close_connection()
```

Conclusion

Congratulations! You've successfully implemented a hybrid authentication system that combines symmetric and asymmetric encryption methods. This approach provides enhanced security by securely managing the symmetric key with asymmetric encryption while efficiently encrypting and decrypting user passwords using symmetric encryption. Remember to handle sensitive data securely, use HTTPS for communication, and follow best practices to ensure the security of your applications.