CS-3002 INFORMATION SECURITY

Assignment 2

Group Members:

Minam Faisal (21i-1901)

Momenah Saif (21i-1909)

An Implementation of Zero Knowledge Authentication

Introduction:

NARWHAL is a revolutionary zero-knowledge authentication system that is presented in the paper "NARWHAL: An Implementation of Zero Knowledge Authentication". NARWHAL uses a challenge-response architecture to improve the security of traditional hashed-password systems. By ensuring that password information is not transferred, this strategy guards against potential compromise and interception. NARWHAL also places a high priority on user privacy and provides tools for users to confirm that their passwords are stored securely.

NARWHAL is committed to enhancing security protocols and addressing typical vulnerabilities seen in traditional login systems, even in the face of identified potential vulnerabilities. Using novel authentication techniques, NARWHAL hopes to create a strong foundation that places an emphasis on user security and privacy in the digital domain.

Summary of Implementation:

Setup:

When the server first loads the authentication system, it generates a public key for itself. This consists of a cryptographic group G as well as some element g0 of G.

Registration:

The client receives the server's public key when a user logs in with their username and password. Y = g0^x is computed by the client and sent to the server along with the pair (username, Y). Let x be the hash of the password. Specifically, the password is not sent to the server during registration and given the complexity of the discrete logarithm issue in G, the server is not provided with any information that would be helpful in figuring out x. Y shall be designated as the user's public key.

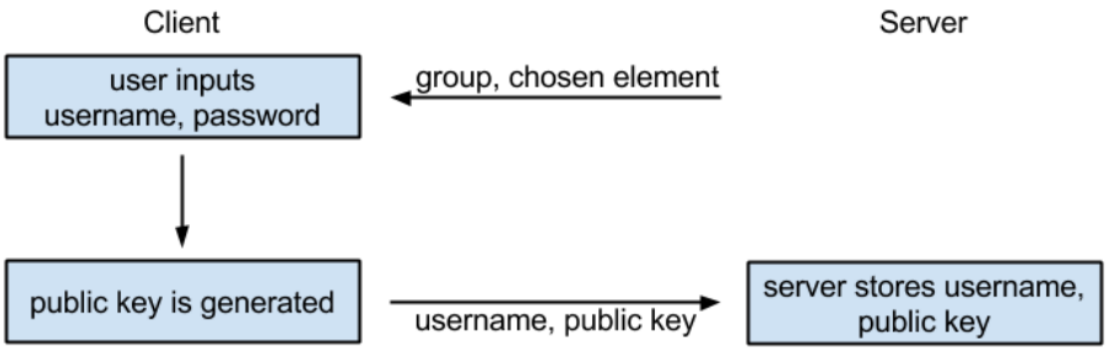

Figure 1. Registration procedure.

Authentication:

For the user to authenticate, the server must first create, store, and transmit a challenge to the user in the form of a random integer that is associated with their username. After the user enters a password, the client determines the following numbers:

- Y = g0^x
- r, a secret random number unique to this authentication attempt
- x = hash(password)
  Z = r − cx; T = g r 0
- c = hash (Y ||T||a)

The pair (c, z), which we refer to as the user's response, is then sent by the client to the server. The user's identity can be verified by the server by verifying the computation of c alone. Given that it is aware of both Y and a,. If the value of hash (Y ||T 0 ||a) matches the value of c provided by the user, the server accepts the authentication attempt.

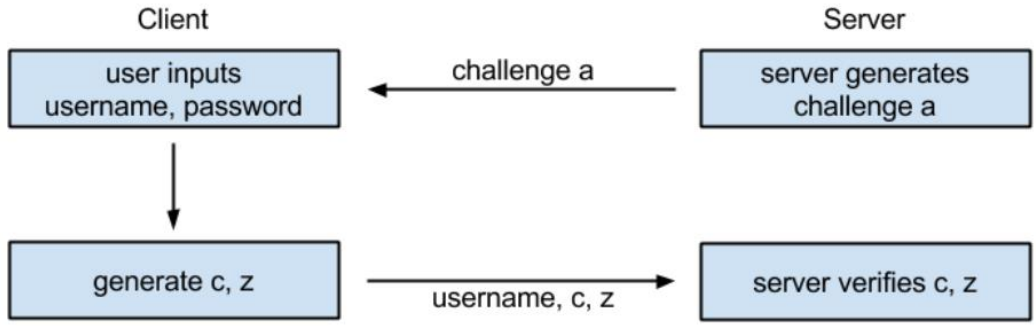

Figure 2. Authentication Procedure.

Implementation in Code:

- We used Python language for implementation.

- The AuthenticationSystem class defines the setup, registration, and authentication processes.

```
import hashlib
import random

class AuthenticationSystem:
```

```
def main():

    auth_system = AuthenticationSystem()
    print("\033[1;35;40m" + "*******************************************************************************\n" + "\033[0m")
    print("\033[1;32;40m" + "          An Implementation of Zero Knowledge Authentication\n" + "\033[0m")
    print("\033[1;35;40m" + "*******************************************************************************\n" + "\033[0m")
```

Setup:

- The __init__ method within the AuthenticationSystem class is responsible for setup. It initializes the attributes such as users, G, and g0. Our mathematical calculations will take place on that group.

```
def __init__(self):
    self.users = {}
    self.G = [random.randint(0, 9) for _ in range(155)]
    self.g0 = [random.randint(0, 9) for _ in range(155)]
```

- In main (), we will just call them.

```
print("\n\033[1;37;42mServer:\033[0m")
print("\n\033[1;36;40mG: " + str(auth_system.G) + "\033[0m")
print("\n\033[1;33;40mg0: " + str(auth_system.g0) + "\033[0m")
print("\n")
```

Registration:

- The register_user method is responsible for user registration. After computing G and g0 Menu will be prompt to user.

```
while True:
    print("\033[1;35;40m" + "*****************************************\n" + "\033[0m")
    print("\033[1;32;40m" + "                 Menu \n" + "\033[0m")
    print("\033[1;35;40m" + "*****************************************\n" + "\033[0m")
    action = input("\033[1;32;40m**Enter any one of following**\n1. 'R' for registration\n2. 'A' for authentication\n3. 'Q' to quit\n\033[0m")
```

- If user chooses “R”, then it takes a username and password.

```
if action == 'R':
    print("\033[1;35;40m" + "****************************************\n" + "\033[0m")
    print("\033[1;32;40m" + "                Crendentails \n" + "\033[0m")
    print("\033[1;35;40m" + "****************************************\n" + "\033[0m")
    username = input("\033[1;33;40mEnter username: \033[0m")
    while True:
          password = input("\033[1;32;40mEnter password:\n \033[0m")
          if len(password) < 8:
              print("\033[1;33mPassword must be at least 8 characters long. Please re-enter.\033[0m")

          else:
               break
    auth_system.register_user(username, password)
```

- Function that responsible for registration.

```
def register_user(self, username, password):
    # Check if the username already exists
    if username in self.users:
        print("\033[1;31mError: Username already exists!\033[0m")

        return
```

- Function first calculates the hashes with sha256 of password entered by user.

```
password_hash = hashlib.sha256(password.encode()).hexdigest()
```

- Convert the password hash to int to computes the public key “Y”.

```
# Compute the public key Y = g0^x
x = int(password_hash, 16)  # Convert the hash to integer
Y = [pow(g, x, len(self.G)) for g in self.g0]  # Computing g0^x mod |G|
```

- Stores user information and prints the registration details.

```
# Store the user's information
self.users[username] = {'Y': Y, 'login_attempts': 0}
print("\033[1;33;40m" + "********************************\n" + "\033[0m")
print("\033[1;40m User registered successfully!\n\033[0m")
print("\033[1;33;40m" + "********************************\n" + "\033[0m")

print("\033[1;35;40m" + "****************************************\n" + "\033[0m")
print("\033[1;32;40m" + "            Password HASH \n" + "\033[0m")
print("\033[1;35;40m" + "****************************************\n" + "\033[0m")
print("\033[1;34mHash:\033[0m", "\033[1;32m", password_hash, "\033[0m")
print("\n")


print("\033[1;35;40m" + "****************************************\n" + "\033[0m")
print("\033[1;32;40m" + "             Public Key \n" + "\033[0m")
print("\033[1;35;40m" + "****************************************\n" + "\033[0m")
print("\033[1;34mPublic key:\033[0m", "\033[1;32m", Y, "\033[0m")
print("\n")
```

Authentication:

- The authenticate_user method is responsible for user authentication. After registration process it will again prompt the menu to user.

```
while True:
    print("\033[1;35;40m" + "****************************************\n" + "\033[0m")
    print("\033[1;32;40m" + "                Menu \n" + "\033[0m")
    print("\033[1;35;40m" + "****************************************\n" + "\033[0m")
    action = input("\033[1;32;40m**Enter any one of following**\n1. 'R' for registration\n2. 'A' for authentication\n3. 'Q' to quit\n\033[0m")
```

- If user enter “A”, it will take username and password and pass it to the class where authentication function will authenticate the user.

```
elif action == 'A':
    print("\033[1;35;40m" + "****************************************\n" + "\033[0m")
    print("\033[1;32;40m" + "              Authentication \n" + "\033[0m")
    print("\033[1;35;40m" + "****************************************\n" + "\033[0m")

    username = input("\033[1;34mEnter username for authentication: \033[0m")
    password = input("\033[1;34mEnter password for authentication: \033[0m")


    auth_system.authenticate_user(username, password)
```

- It verifies the username, checks if the user is locked.

```
def authenticate_user(self, username, password):
    if username in self.users:
        if self.users[username].get('locked', False):
            print("\033[1;35;40m" + "****************************************\n" + "\033[0m")
            print("\033[1;32;40m" + "              LOGIN FAILED \n" + "\033[0m")
            print("\033[1;35;40m" + "****************************************\n" + "\033[0m")
            print("User is locked. Contact administrator.")
            return
```

- Calculates password hash, convert it t integer and generates public key.

```
password_hash = hashlib.sha256(password.encode()).hexdigest()
x = int(password_hash, 16)
Y = self.users[username]['Y']
```

- Then calculate “r”, which is a random number specific to authentication attempt.

```
r = random.randint(10**(155-1), 10**155-1)
print("\033[1;35;40m" + "********************************************************************************\n" + "\033[0m")
print("\033[1;32;40m" + "      r, a private random number specific to this authentication attempt          \n" + "\033[0m")
print("\033[1;35;40m" + "********************************************************************************\n" + "\033[0m")
print("\033[1;34mr: \033[0m", "\033[1;33m", r, "\033[0m", "\n")
```

- “T” is used as part of the challenge-response mechanism to ensure security during the authentication process.

```
# Compute T = g0^r
T = [pow(g, r, len(self.G)) for g in self.g0]
```

- During authentication, the server generates a random challenge “a”, which is then used in combination with the user's public key “Y” and the value “T” to compute a hash “c”.

```
# Generate a random challenge 'a'
a = random.randint(10**(155-1), 10**155-1)
print("\033[1;35;40m" + "************************************************\n" + "\033[0m")
print("\033[1;32;40m" + "          a, a random challenge           \n" + "\033[0m")
print("\033[1;35;40m" + "************************************************\n" + "\033[0m")
print("\033[1;34ma: \033[0m", "\033[1;33m", a, "\033[0m", "\n")
```

- “c” is used as part of the authentication process to verify the integrity of the exchanged data and prevent replay attacks.

```
# Compute c = hash(Y || T || a)
concatenated_data = "".join(map(str, Y + T + [a]))
c = hashlib.sha256(concatenated_data.encode()).hexdigest()
```

- “z” is a value computed as z=r−cx, where r is the same random private number mentioned earlier, “c” is the hash of certain concatenated data (including the public key “Y”, the value “T”, and a random challenge “a”), and “x” is derived from the user's password hash. It’s used as part of the authentication process to compute a response to the server's challenge.

```
# Compute z = r - cx
z = (r - int(c, 16) * x)
```

- Now simply send user side authentication (user_response) to server for server-side authentication.

```
# Send the pair (c, z) to the server
user_response = (c, z)
```

- The server retrieves the public key Y associated with the username from storage. It then calculates: T' = Yc * g0 z mod p hash (Y || T' || challenge). If hash (Y || T' || a) matches the value of c provided by the user, the server accepts the authentication attempt.

```
# Server checks if hash(Y || T' || a) matches c
T_prime = [(pow(Y[i], int(c, 16), len(self.G)) * pow(self.g0[i], z, len(self.G))) % len(self.G) for i in range(len(Y))]
concatenated_data_prime = "".join(map(str, Y + T_prime + [a]))
c_prime = hashlib.sha256(concatenated_data_prime.encode()).hexdigest()
print("\033[1;35;40m" + "************************************************************************************\n" + "\033[0m")
print("\033[1;32;40m" + "                          z = r - cx            \n" + "\033[0m")
print("\033[1;35;40m" + "************************************************************************************\n" + "\033[0m")
print("\033[1;34mz: \033[0m", "\033[1;33m", z, "\033[0m", "\n")


print("\033[1;35;40m" + "************************************************************************************\n" + "\033[0m")
print("\033[1;32;40m" + "                        T = g0^r            \n" + "\033[0m")
print("\033[1;35;40m" + "************************************************************************************\n" + "\033[0m")
print("\033[1;34mT_prime: \033[0m", "\033[1;33m", T_prime, "\033[0m", "\n")


print("\033[1;35;40m" + "************************************************************************************\n" + "\033[0m")
print("\033[1;32;40m" + "                          c = hash(Y ||T||a)            \n" + "\033[0m")
print("\033[1;35;40m" + "************************************************************************************\n" + "\033[0m")
print("\033[1;34mc_prime: \033[0m", "\033[1;33m", c_prime, "\033[0m", "\n")
```

- And finally prompts the successful authentication by user message on terminal.

```
if c_prime == c:
    print("\033[1;35;40m" + "**********************************************************************\n" + "\033[0m")
    print("\033[1;32m            Authentication successful for user '{}'!\033[0m\n".format(username))
    print("\033[1;35;40m" + "**********************************************************************\n" + "\033[0m")
```

- If password is wrong, then it prompt authentication is failed message.

```
  else:
      print("\033[1;31mAuthentication failed. Please try again.\033[0m")
```

- If login attempts will greater then 3, then the user will be locked from other attempts.

```
if self.users[username]['login_attempts'] > 3:
    print("\033[1;35;40m" + "****************************************\n" + "\033[0m")
    print("\033[1;31;40m" + "            USER IS LOCKED \n" + "\033[0m")
    print("\033[1;35;40m" + "****************************************\n" + "\033[0m")
    del self.users[username]
    return
```

Conclusion:

In summary, the Zero-Knowledge Authentication system, as demonstrated in the Python code, presents a reliable approach to user authentication. By maintaining confidentiality and preventing the exposure of sensitive information during the authentication process, it establishes a strong defence against potential security threats in conventional login systems. Furthermore, the implementation of this system underscores the importance of balancing security and usability in authentication mechanisms. While focusing on enhancing security measures, it also strives to optimize user experience by minimizing complexities in the authentication process. Through continual refinement and adaptation, such systems can evolve to meet the ever-changing landscape of cybersecurity threats, safeguarding user data and privacy effectively.

THE END...