**By :**

**Omar Ahmed Badrawy – Id: 10005368**

**Ahmed Alaaeldin – Id: 10007101**

**Zero-Knowledge Proof-Based Client-Server Authentication System**

**ICS 505 Cryptography**

**Home Assignment 2**

**Dr. Muhammad Hataba**

*muhammad.hataba@giu-uni.de*

**TA. John Ehab**

*john.ehab@giu-uni.de*

**Submission Link:** https://forms.gle/3pKeh73323NAtpx76

---

**Table of Contents**

Introduction

Objectives

System Design

Client-Server Architecture

Zero-Knowledge Proof Integration

Implementation

Client-Side Implementation

Server-Side Implementation

Circom and Snarkjs Integration

Code Explanation

Client Code (`client.py`)

Server Code (`server.py`)

Testing and Results

Execution Outputs

Performance Metrics

Metrics Collection

CPU and RAM Utilization

Packet Propagation and Processing Time

Analysis

Functionality Assessment

Performance Evaluation

Security Evaluation

Conclusion

Future Work

References

Appendices

Appendix A: Client Code

Appendix B: Server Code

Appendix C: Circom Circuit

---

**1. Introduction**

Zero-Knowledge Proofs (ZKPs) are advanced cryptographic protocols that enable one party to prove to another that they know a particular piece of information without revealing the information itself. This technology is pivotal in enhancing authentication systems by ensuring secure verification while preserving user privacy. This report details the development and implementation of a ZKP-based one-factor authentication system using a client-server architecture in Python, integrating Circom and snarkjs for proof generation and verification.

---

**2. Objectives**

The primary objectives of this project are:

**Implement a Client-Server Architecture:** Develop a Python-based client and server that communicate securely using socket programming.

**Integrate Zero-Knowledge Proofs:** Utilize Circom and snarkjs to generate and verify ZKP proofs, ensuring that user credentials (username and password) are authenticated without transmitting the password in plaintext.

**Handle Enrollment and Authentication:** Implement functionalities for user sign-up (enrollment) and login (authentication) using ZKPs.

**Collect and Analyze Metrics:** Monitor and evaluate CPU and RAM utilization, packet propagation, and processing times to assess the system's efficiency and usability.

---

**3. System Design**

**Client-Server Architecture**

The system consists of two primary components:

**Client (client.py):** Handles user interactions, generates ZKP proofs during sign-up and login, and communicates with the server.

**Server (server.py):** Listens for incoming client connections, verifies ZKP proofs, and manages user data securely.

**Zero-Knowledge Proof Integration**

The integration leverages **Circom**, a domain-specific language for writing ZKP circuits, and **snarkjs**, a JavaScript library for generating and verifying proofs. The process involves:

**Circuit Definition:** Creating a Circom circuit that outlines the relationship between inputs (username and password) and the ZKP proof.

**Proof Generation:** Using Circom and snarkjs in the client to generate proofs based on user inputs.

**Proof Verification:** The server employs snarkjs to verify the received proofs without accessing the actual passwords.

---

**4. Implementation**

**Client-Side Implementation**

The client application is responsible for:

**User Interaction:** Allowing users to sign up and log in by entering their credentials.

**Proof Generation:** Utilizing Circom and snarkjs to generate ZKP proofs based on user inputs.

**Secure Communication:** Sending proofs and public signals to the server via socket connections.

**Server-Side Implementation**

The server application handles:

**Listening for Connections:** Awaiting client requests on a specified port.

**Proof Verification:** Using snarkjs to verify the integrity of received ZKP proofs.

**User Data Management:** Storing and retrieving user commitments and proofs securely.

**Circom and Snarkjs Integration**

Integration steps include:

**Circuit Definition:** Creating password\_verify.circom to define the ZKP circuit for password verification.

**Circuit Compilation:** Compiling the circuit using Circom to generate the necessary artifacts (.r1cs, .wasm, .sym).

**Trusted Setup:** Performing the trusted setup phase with snarkjs to generate proving and verification keys.

**Proof Generation and Verification:** Automating proof generation in the client and verification in the server using snarkjs commands.

---

**5. Code Explanation**

**Client Code (client.py)**

A black background with a yellow star

Description automatically generated

**Key Components:**

**Encryption Helpers:** Functions to encrypt and decrypt sensitive data stored locally.

**ZKP Operations:** Functions to compute modular exponentiation and multiplicative operations essential for ZKP calculations.

**Circom Integration:** Automates the generation of ZKP proofs by interfacing with Circom and snarkjs through subprocess calls.

**Client Operations:** Handles user sign-up and login by generating proofs, sending requests to the server, and storing secrets securely.

 **Metrics Collection:** Measures the time taken for operations and resource utilization.

**Server Code (server.py)**

**A screen shot of a computer

Description automatically generatedKey Components:**

**JSON Storage:** Functions to load and save user records persistently.

**ZKP Verification:** Utilizes snarkjs to verify received ZKP proofs, ensuring that authentication is secure.

**Handle Connection:** Manages incoming client connections, processes sign-up and login requests, and records performance metrics.

**Metrics Collection:** Tracks the time and memory consumption for each request to evaluate performance.

---

**6. Testing and Results**

**Execution Outputs**

**Server Output:**

A screen shot of a computer program

Description automatically generated**Client Output:**

A screenshot of a computer

Description automatically generated

**Performance Metrics**

**Sign-Up Operation:**

**Time Taken:** 0.006 seconds

**Memory Allocation Change:** 32,768 bytes

**Login Operation:**

**Time Taken:** 0.001 seconds

**Memory Allocation Change:** 8,192 bytes

---

**7. Metrics Collection**

**CPU and RAM Utilization**

Both the client and server utilize the psutil library to monitor CPU and RAM usage during operations. Metrics are printed to the console and can be logged for detailed analysis.

**Example:**

A screen shot of a computer

Description automatically generated

**Packet Propagation and Processing Time**

The system measures the time taken from sending a request to receiving a response using Python's time module. This helps in evaluating the communication latency and processing efficiency.

**Example:**

A computer screen shot of a program code

Description automatically generated---

**8. Analysis**

**Functionality Assessment**

**Sign-Up and Login Operations:** The client successfully generates and sends ZKP proofs during sign-up and login. The server accurately verifies these proofs, ensuring secure authentication without transmitting plaintext passwords.

**Data Persistence:** User commitments and proofs are stored securely on the server, ensuring data integrity and confidentiality.

**Performance Evaluation**

**Efficiency:** The system processes sign-up and login requests rapidly, with minimal latency and low memory overhead, indicating an efficient implementation.

**Scalability:** While initial tests show promising performance, further testing with multiple concurrent users is recommended to assess scalability.

**Security Evaluation**

**Zero-Knowledge Proofs:** The integration ensures that user passwords are never exposed during authentication, adhering to the principles of ZKPs.

**Data Encryption:** Client-side secrets are encrypted using AES-128, providing an additional layer of security.

**Error Handling:** Robust error handling mechanisms are in place to manage invalid proofs, connection issues, and other potential errors gracefully.

---

**9. Conclusion**

The implementation of a Zero-Knowledge Proof-based client-server authentication system meets the project objectives by ensuring secure and private user authentication. The integration of Circom and snarkjs facilitates robust proof generation and verification, while the client-server architecture enables efficient communication and data management. Performance metrics indicate that the system operates with high efficiency and low resource consumption. Future enhancements can focus on scalability, comprehensive security audits, and user experience improvements.

---

---

**10. References**

**Circom Documentation:** https://docs.circom.io/

**Snarkjs GitHub Repository:** https://github.com/iden3/snarkjs

**Zero-Knowledge Proofs Overview:** https://en.wikipedia.org/wiki/Zero-knowledge\_proof

**psutil Documentation:** https://psutil.readthedocs.io/en/latest/

**Cryptography with Python:** https://cryptography.io/en/latest/

---

**11. Appendices**

**Appendix A: Client Code**

A black screen with green text

Description automatically generated**Appendix B: Server Code**

A screen shot of a computer

Description automatically generated**Appendix C: Circom Circuit**

A screenshot of a computer program

Description automatically generated---

**Final Notes**

This report provides a comprehensive overview of the development, implementation, and evaluation of a Zero-Knowledge Proof-based authentication system. By adhering to cryptographic best practices and integrating robust proof generation and verification mechanisms, the system ensures secure and private user authentication suitable for practical applications.