**Assignment Week 2: Understanding and Applying Proof Systems**

**Table 1. List of students who participated in the assignment**

| No. | Fullname | Note |
| --- | --- | --- |
| 1 | Trần Nhơn Nhật |  |
| 2 | Nguyễn Trung Nguyên | Leader |
| 3 | Phạm Thanh Doãn |  |
| 4 | Phan Minh Quân |  |
| 5 | Ngô Tấn Phước |  |
| 6 | Nguyễn Viết Khang |  |
| 7 | Trần Công Tường |  |
| 8 | Trần Ngọc Thành |  |

1. **What is a proof system?**

- A proof system is a mathematical or logical framework used to verify the correctness of statements, based on inference rules and axioms to construct systematic proofs.

**- Key properties**

Here are the key properties of a proof system, presented concisely:

+ Completeness: Every true statement can be proven.

+ Soundness: No false statement can be proven.

+ Zero-knowledge: The proof does not reveal additional information beyond the truth of the statement.

**- Common categories of Proof Systems: Interactive vs. Non-interactive, ZKP**

+ Interactive Proof Systems: Involve multiple rounds of communication between the prover and the verifier to confirm the validity of a statement.

+ Non-Interactive Proof Systems: The prover sends a single proof that the verifier can independently verify without any further interaction.

+ Zero-Knowledge Proofs (ZKP): A method of convincing the verifier of the truth of a statement without revealing any additional information; it can be either interactive or non-interactive.

**- Example:**

+ Groth16: An efficient zk-SNARK that produces short proofs and fast verification but requires a trusted setup specific to each application.

+ PLONK: A flexible zk-SNARK that uses a universal trusted setup for multiple applications and supports custom gates for optimization.

+ STARK: A ZKP system that does not require a trusted setup, is post-quantum secure, but generates larger proofs and has slower verification.

1. **Applying a Proof System to Solve a Real-World Problem**

#### **2.1.1. Describe the Problem**

**Context:**Enterprises use Role-Based Access Control (RBAC) to grant access permissions based on roles.

**Problem(s):**Users need to authenticate their roles without revealing their identities, especially in high-privacy environments (e.g., HR departments, cybersecurity teams, etc.).

Why Use a Proof System?

Zero-Knowledge Proofs (ZKP) allow proving access rights without disclosing identifiable information, solving challenges related to privacy, verifiability, and integrity.

#### **2.1.2. Introduce the Solution with a Proof System**

**Solution:**The user generates a zero-knowledge proof demonstrating they possess a credential confirming their authorized role.

**How ZKP Improves the System:**

* **Privacy:** No identity is revealed.
* **Verifiability:** The server can validate the proof.
* **Integrity:** Credentials cannot be forged without the secret.

#### **2.1.3. Detailed Design of the Solution**

**a. Problem Modeling**

* **Participants:**
  + *User (Prover)*: Holds a credential containing their role.
  + *Access Server (Verifier)*: Verifies the proof.
  + *Merkle Tree Server*: Stores the access permission tree.
* **Data Flow (textual):**User → Generates proof (from role hash & Merkle path) → Sends to Verifier → Verifier validates → Grants/denies access

**b. Architecture of the Proposed System**

* **Prover:** Possesses a credential (hashed role + Merkle proof).  
  **Verifier:** Checks if the proof belongs to a valid role in the Merkle tree.
* **Merkle Tree:** The root contains all authorized roles.
* **Flow:**
  1. User generates proof using a circuit.
  2. Sends proof to the server.
  3. Server verifies without knowing the specific role.

**c. Techniques and Tools to Be Used**

* **ZK Technique:** zk-SNARK or PLONK
* **Circuit Framework:** Circom
* **Merkle Tree Tool:** SnarkJS + circomlib
* **(Optionally)** Noir + nargo for more readable syntax.

**d. Implementation Plan**

* **Week 1:** Analyze roles, build a sample Merkle tree.
* **Week 2:** Write ZKP circuit (verify hash + Merkle inclusion).
* **Week 3:** Implement verifier + proof generator.
* **Week 4:** Connect to a web demo or CLI test.
* **Week 5:**Testing & and start doing documentation.

**e. Task Assignment for team working**

* **Member A:** Circuit design & proof generation
* **Member B:** Frontend (authentication interface)/Backend
* **Member C:** Verifier logic + Merkle tree
* **Member D:** Documentation, testing, reporting.