Zero Knowledge and IPFS Based Know Your Customer

IT465 Course Project

Sachin Prasanna : 211IT058
Abhayjit Singh Gulati : 211IT085

Under the Guidance of
Dr. Bhawana Rudra, Assistant Professor

Dept. of Information Technology
National Institute of Technology Karnataka, Surathkal

November 7, 2024

Overview

Introduction

- **KYC Importance:** Essential for institutions to verify client identities, ensuring regulatory compliance.
- **Challenges:** Traditional KYC methods incur high costs, pose privacy risks, and rely on centralized systems.
- **Blockchain Solution:** Offers decentralization and immutability, enhancing security and reducing operational costs.
- **Zero-Knowledge Proofs (ZKPs):** Enable identity verification without revealing sensitive information, suitable for privacy-sensitive environments.
- **Research Potential:** ZKPs' broad applicability and minimal security assumptions make them a promising solution for efficient, secure, and private KYC systems.

Literature Survey

- **Malhotra et al. :** Proposed a solution for implementing KYC using a private blockchain where a client's identity is verified only once, and the verified information is securely stored for future access by financial institutions.
- **Yadav et al. :** Developed an approach where authenticated KYC details are added to the blockchain using Ethereum API and Solidity. Users are notified upon verification and can easily apply to banks.
- **Partala et al. :** Understood Zero-Knowledge Proof (ZKP) which was applied to confidential transactions and private smart contracts on blockchains, highlighting the benefits of privacy.
- **Li et al. :** Proposed a decentralized, privacy-preserving architecture integrating ZKPs into blockchain-based traffic management systems to ensure both data integrity and privacy.
- **General KYC Challenges:** Addressed by the need for blockchain technology in KYC processes to overcome issues such as human error, lack of skilled personnel, and time delays.

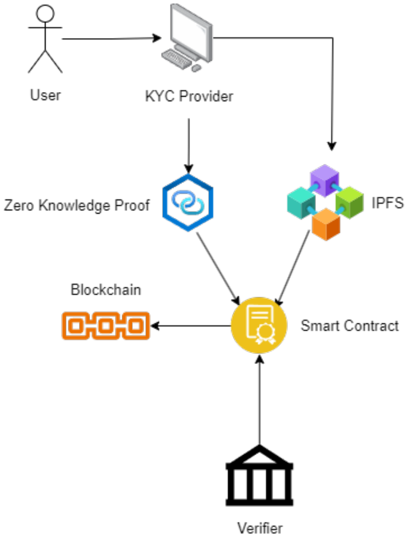
Problem Statement

In a traditional KYC (Know Your Customer) process, users need to provide sensitive personal information such as name, address, and date of birth to service providers like banks, insurance companies, or government agencies. This data is typically stored in centralized databases, raising concerns about privacy, data breaches, and unauthorized access.
Furthermore, service providers often require specific information (e.g., verifying that a person is over 18), but not the full details of the person's identity or address. The challenge is to create a system that allows users to prove certain facts about themselves (e.g., age) without revealing sensitive information.

Objectives

- **Data Privacy and Security:**
  - Implement a decentralized KYC system using Zero-Knowledge Proofs (ZKPs).
  - Ensure users can prove attributes without revealing full personal details.
  - Store sensitive data on IPFS for privacy and protection.
- **Decentralized and Immutable Storage:**
  - Use blockchain to record cryptographic proofs of KYC updates.
  - Ensure no centralized authority can alter records.
  - Create a tamper-proof, auditable ledger for KYC compliance.
- **Efficient Verifiability:**
  - Enable service providers to verify user attributes via ZKPs.
  - Reduce manual checks through secure, automated blockchain verification.
  - Maintain privacy while ensuring fast and efficient verifiability.

Architecture



Technology Stack

- **Solidity**: A programming language used for writing smart contracts on the Ethereum blockchain, enabling secure and decentralized logic for KYC processes.
- **Python**: A versatile programming language used for writing scripts that interact with the blockchain and execute Zero-Knowledge Proofs (ZKPs) for verification without revealing personal data. It will also be used to take input from the user for the KYC.
- **Ganache**: A personal Ethereum blockchain for local development that allows for the testing of smart contracts and simulates the behavior of a real blockchain environment.
- **Truffle Suite**: To compile and deploy the smart contract.
- **Pinata API**: A service that simplifies the process of storing and managing files on the InterPlanetary File System (IPFS), allowing for the secure and decentralized storage of KYC data.
- **Flask** : Serves as the primary interface for handling KYC registration, updates, and verification through an Ethereum-based smart contract. The application integrates various components, including IPFS, a smart contract and zero-knowledge proof (ZKP).

Implementation Overview

- **Smart Contract Overview:**
  - KYC smart contract enables secure and automated client information management for KYC verification.
  - Stores essential details: report URI, KYC expiration date, and zero-knowledge proof (ZKP) for age verification.
  - Includes functions for registering, updating, and verifying client information based on expiration.
  - Event logs for tracking registrations, updates, and expiring reports.
- **Zero-Knowledge Proof (ZKP):**
  - Verifies age without revealing specific details.
  - Uses an identifier to generate a unique, non-reversible 32-character ”proof” based on age threshold (e.g., 18 years).
  - Balances verification with data privacy, ensuring anonymity.
- **Interplanetary File System (IPFS):**
  - IPFS provides decentralized storage and retrieval of KYC data.
  - Client information structured in JSON, uploaded via Pinata API.
  - IPFS CID (Content Identifier) serves as a secure reference, ensuring data integrity and accessibility.

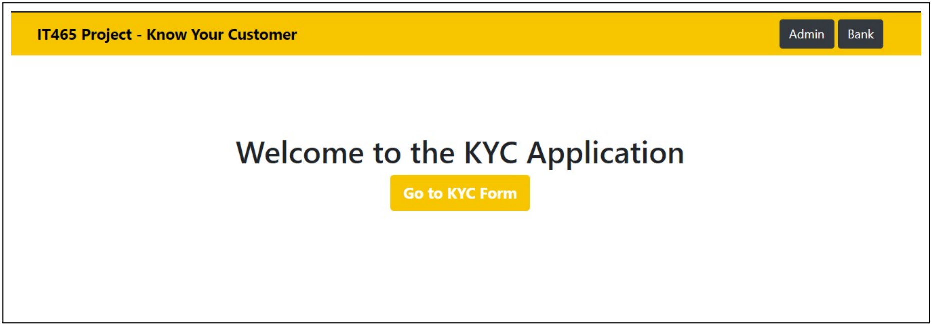
Flask Application

- **Application Initialization:**
  - Initializes Flask app, loads KYC smart contract through `initContract()` for Ethereum interaction.
- **Helper Functions:**
  - `calculate_age(dob)`: Calculates user's age from DOB (dd/mm/yyyy).
- **KYC Registration Route:**
  1. Data Collection: Personal information such as name, DOB, email, etc.
  2. ZKP Proof Generation: Age verification without exposing exact age.
  3. Data Conversion & Storage in IPFS: Structured JSON uploaded to IPFS via Pinata.
  4. Smart Contract Interaction: Registers KYC data with IPFS URI and ZKP.

Flask Application (Continued)

- **KYC Update Route:**
  - Allows updating KYC data with new IPFS URI and ZKP.
- **Admin and Bank Routes:**
  - `Admin Page`: Allows admin to verify KYC records, retrieve client counts, and view client details.
  - `Bank Page`: Allows banks to verify age eligibility based on ZKP.
- **Admin and Bank Authentication:**
  - `Admin Login:` manages role-based access control with passwords for administrators and bank officials.
- **Blockchain Transactions:**
  - Includes functions like `registerKYC`, `updateKYC`, `checkValidity`, `getClientCount`, and `Clientdatabase`.
- **IPFS & ZKP Integration:**
  - IPFS for secure, decentralized storage; ZKP for privacy-preserving age verification.

Demonstration


IT465 Project - Know Your Customer
Admin
Bank
Welcome to the KYC Application
Go to KYC Form

Conclusions

- The project successfully implemented a KYC smart contract that includes functionalities for client registration, KYC validation, and age verification using Zero Knowledge Proof (ZKP).
- A Flask-based Python application was developed to gather KYC data from users. This data is stored in IPFS via Pinata, while only essential details like the ZKP and IPFS hash are stored on-chain, ensuring a lightweight and secure blockchain.
- An "Admin Mode" feature allows administrators to view user information, check KYC validity, and monitor the total number of registered clients.
- The "Bank Mode" enables banks to verify a user's age (above 18) using the user ID and ZKP stored on-chain. Banks must authenticate themselves via password before verifying user age.

Thank You