**Proposed Method :**

**Present Solution**

The solution is a permission-based KYC Blockchain network, with trusted parties involved in the KYC verification process. For example, the customer submits the KYC documents such as passport copy, driving license copy, address proof, and other documents as proof to open a new bank account or to perform any new financial activities in, say, Bank A.

Bank A verifies and stores it in the blockchain distributed ledgers. The system will create a KYC ID for registered KYC on the network. Now, when the customer opens a new bank account or to apply for a loan in another bank, say, Bank B, the customer need not submit the same documents to Bank B again. Instead, he can pass on the KYC ID to Bank B, and Bank B can retrieve the data from the distributed ledger, which is already verified by Bank A. For all this process to happen, Bank B must join as the participant organization of the permission-based KYC network and must accept the endorsement policy of Bank A.

In the KYC verification process, Blockchain technology is implemented in two ways:

1. **Intra Bank**

To conduct the KYC verification process, blockchain-based KYC makes the process hassle-free for the customer and ensures trust between departments and branches.

**2. Inter Bank**

Here, to conduct the KYC verification process, Bank A acts as the initiating bank and performs an initial KYC documents verification process for a client. The KYC Blockchain network ensures secure inter-Bank documents exchange- enhancing process efficiency. Standardizing KYC processes, and performing near-real-time client validations.

**Technical Details**

The KYC Blockchain solution is designed on a permission-based blockchain platform that is open-source.

The solution makes use of the platform's network, smart contract, and Dapp(Decentralized Application) deployment capabilities.

Smart contracts make it possible to automate KYC in accordance with industry requirements.

It is difficult to develop a high level of confidence in the ecosystem without blockchain.

Permissioned nodes (representing banks) can access, submit, and validate KYC data through the blockchain's distributed database, which intrinsically builds confidence.

The platform employs private keys and encryptions to address client-to-node and node-to-node interactions, hence boosting security.

Privacy over private networks is a feature that tackles the privacy concerns associated with banking KYC sharing.

Every node in eKYC hosts services and runs DApps(Decentralized Apps) on a Java virtual machine.

AMQP is used for inter-node communication (Advanced Message Queuing Protocol).

We have separated the UI technologies by allowing DApps to interface with any UI layer of the eKYC application.

**Technologies Used**

1. Solidity Programming Language :-

Solidity is a statically-typed curly-braces programming language designed for developing smart contracts that run on [Ethereum](https://ethereum.org/en/).

Solidity is a contract-oriented, high-level programming language for implementing smart contracts. Solidity is highly influenced by C++, Python and JavaScript and has been designed to target the Ethereum Virtual Machine (EVM).

Solidity is statically typed, supports inheritance, libraries and complex user-defined types programming language.

1. The Ethereum Virtual Machine (EVM) :-

The Ethereum Virtual Machine, also known as EVM, is the runtime environment for smart contracts in Ethereum. The Ethereum Virtual Machine focuses on providing security and executing untrusted code by computers all over the world.

The EVM specialised in preventing Denial-of-service attacks and ensures that programs do not have access to each other's state, ensuring communication can be established without any potential interference.

The Ethereum Virtual Machine has been designed to serve as a runtime environment for smart contracts based on Ethereum.

1. Smart Contracts :-

A smart contract is a computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract. Smart contracts allow the performance of credible transactions without third parties. These transactions are trackable and irreversible.

The concept of smart contracts was first proposed by Nick Szabo in 1994. Szabo is a legal scholar and cryptographer known for laying the groundwork for digital currency.

**Methodology:**

Firstly, We will create a web application for client side where clients can upload KYC details of their customer. This client application contains several web pages other than just uploading clients can also find Kyc-Hashes for customers who had their KYC details on blockchain already. This client side application is also connected to the clients database where they can store secret keys and other information about their customers.

**Client Application**

The client application employs REST API endpoints to upload (POST) and retrieve (GET) documents. It's written in the NodeJS programming language. When it comes to uploading, there are a few things to keep in mind.

* Creating and encoding header, header signature, and payload transactions. (Transaction payloads are made up of binary-encoded data that the validator can't see).
* Creating BatchHeaders, Batches, and Batch Encoding.
* Batches are submitted to the validator.

1. **Uploading Records Of Customer :-**

Records can be uploaded in any format (doc, pdf, jpg, etc.) and can be up to 10 MB in size.

These records are immediately encrypted with the AES symmetric encryption algorithm, and the decryption keys are held in the uploading entity's unique database.

The following information must be provided when a new record is posted to the blockchain:

1. The Organization ID (OI) of the entity to which this document belongs is stored in plain text / unencrypted form in the blockchain and cannot be modified..
2. Document type — this information is stored in plain text / unencrypted form on the blockchain and cannot be modified.
3. A brief description of the document — this data is recorded in plain text / unencrypted form on the blockchain and cannot be modified.
4. The document – which might be in pdf, word, excel, picture, or other format – is saved in the blockchain in AES-encrypted form and is unchangeable.
5. The decryption key is kept in the relevant bank's computer system.

This is what occurs when the above information is provided:

1. The uploaded file's hash is calculated.
2. The file is digitally signed using the uploader bank's private key.
3. AES symmetric encryption is used to encrypt the file.
4. Hexadecimal is used to decode the encrypted data.
5. Hexadecimal is used to transform non-encrypted data.
6. The blockchain gets updated with hexadecimal material.
7. **Getting Uploaded Documents :-**

Using the GET technique to retrieve the data stored on the particular address, the identical address is created from the OI provided by the user.

updatedAddress is formed by collecting user input either from User (search using OI in the network) or from the user's private database, as shown in the following code excerpt (Records uploaded by the user).

Similarly, because of the transaction logic built in the Transaction Processor to upload numerous documents on the same address while updating state with a list of all the uploaded data, splitStringArray splits the data returned from that address (not only the current payload).