e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:04/April-2024 Impact Factor- 7.868 www.irjmets.com

KYC USING BLOCKCHAIN TECHNOLOGY

Sushant Singh[*1], Raj Mishra[*2], Omkar Talwadekar[*3], Rishabh Tiwari[*4], Prof. Chitra Wasnik[*5], Prof. Ujjwala Pagare[*6]

[*1,2,3,4]Under Graduate Student, Dept. Of Computer Science & Engineering (Artificial Intelligence And Machine Learning) Lokmanya Tilak College Of Engineering, Mumbai University, Navi-Mumbai, India.

[*5]Head Of Department, Dept. Of Computer Science & Engineering (Artificial Intelligence And Machine Learning), Navi-Mumbai, India.

[*6]Professor, Dept., Of Computer Science & Engineering(Artificial Intelligence And Machine Learning), Navi-Mumbai, India.

DOI : https://www.doi.org/10.56726/IRJMETS52546

ABSTRACT

Know Your Customer (KYC) procedures are critical in financial services to verify the identity of clients, prevent fraud, and ensure compliance with regulations. However, traditional KYC processes are often time-consuming, costly, and susceptible to errors and fraud. Blockchain technology presents a promising solution to these challenges by providing a decentralized, transparent, and immutable ledger for storing and verifying identity information. This paper explores the integration of blockchain technology into KYC processes to streamline and enhance identity verification procedures. By leveraging blockchain, financial institutions can create a shared network where customer identity information is securely stored and updated in real-time. This distributed ledger ensures data integrity and eliminates the need for multiple, redundant KYC checks across different institutions. Overall, the adoption of blockchain technology in KYC processes has the potential to revolutionize identity verification in the financial industry, leading to increased efficiency, reduced costs, and enhanced security and trustworthiness.

I. INTRODUCTION

The Know Your Customer (KYC) regulations have long been a cornerstone of the financial industry, serving as a crucial tool for verifying the identities of clients and ensuring compliance with anti-money laundering (AML) and counter-terrorism financing (CTF) laws. However, traditional KYC processes are often cumbersome, time-consuming, and prone to inefficiencies, with each financial institution conducting its own redundant checks on customers. Blockchain technology has emerged as a disruptive force in various sectors, offering a decentralized, transparent, and immutable ledger for recording transactions securely. In recent years, the integration of blockchain technology into KYC processes has garnered significant attention for its potential to streamline identity verification, enhance security, and reduce operational costs. The fundamental concept behind using blockchain for KYC lies in creating a shared, distributed network where customer identity information can be securely stored and accessed by authorized parties in real-time. By leveraging blockchain's cryptographic principles and consensus mechanisms, financial institutions can establish a tamper-proof record of customer identities, transactions, and interactions. Moreover, blockchain-enabled KYC solutions have the potential to empower individuals by giving them greater control over their personal data. Through self-sovereign identity models, individuals can manage and share their identity information selectively, thereby mitigating privacy concerns and reducing the risk of data breaches. This introduction sets the stage for exploring the implications of blockchain technology on KYC processes, highlighting its potential to revolutionize identity verification in the financial industry. By examining the key features and benefits of blockchain-based KYC solutions, this paper aims to provide insights into the transformative impact of technology on regulatory compliance and customer onboarding procedures.

II. METHODOLOGY

Identification and Verification Process Design:

The first step in implementing KYC using blockchain involves designing the identification and verification process. This includes determining the required identity documents and information, as well as the verification

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:04/April-2024 Impact Factor- 7.868 www.irjmets.com

methods to be used. The process should comply with regulatory requirements while leveraging the capabilities of blockchain for enhanced security and efficiency.

Blockchain Platform Selection:

Choose a suitable blockchain platform that aligns with the specific requirements of KYC processes. Consider factors such as scalability, security, interoperability, and consensus mechanism. Ethereum, Hyperledger Fabric, and Corda are among the popular blockchain platforms suitable for KYC applications. Here we have used Ganache.

Smart Contract Development:

Develop smart contracts to automate various aspects of the KYC process, including identity verification, document validation, and consent management. Smart contracts help enforce rules and conditions programmatically, ensuring transparency and immutability of data stored on the blockchain**.**

Data Encryption and Storage:

Implement robust encryption mechanisms to protect sensitive customer data stored on the blockchain. Utilize encryption algorithms such as AES (Advanced Encryption Standard) to secure data at rest and in transit. Determine the appropriate data storage model, whether on-chain, off-chain, or a combination, considering factors like data privacy and regulatory compliance.

Consensus Mechanism Implementation:

Choose an appropriate consensus mechanism to validate and add transactions to the blockchain. Consensus mechanisms such as Proof of Work (PoW), Proof of Stake (PoS), or Practical Byzantine Fault Tolerance (PBFT) ensure the integrity and consensus of the distributed ledger, thereby enhancing the trustworthiness of KYC data.

Integration with External Systems:

Integrate the blockchain-based KYC solution with external systems such as identity verification providers, regulatory databases, and financial institutions' existing KYC platforms. API integration enables seamless data exchange and interoperability between different systems, facilitating a more comprehensive KYC process.

User Interface Development:

Design user-friendly interfaces for both customers and financial institution personnel to interact with the blockchain-based KYC system. Implement intuitive workflows for onboarding new customers, submitting identity documents, and granting consent for data sharing. User interfaces should prioritize security, privacy, and accessibility.

Testing and Deployment:

Conduct thorough testing of the blockchain-based KYC solution to ensure its functionality, security, and compliance with regulatory requirements. Perform testing for scalability, performance, and resilience to simulated attacks. Once testing is complete, deploy the solution in a production environment, ensuring proper monitoring and maintenance.

Regulatory Compliance and Governance:

Establish robust governance mechanisms to ensure compliance with regulatory requirements such as GDPR, AML, and CFT regulations. Define roles and responsibilities for managing and maintaining the blockchain-based KYC system, including data access controls, audit trails, and reporting mechanisms.

Continuous Improvement and Updates:

Continuously monitor the performance and effectiveness of the blockchain-based KYC solution and incorporate feedback from stakeholders to drive continuous improvement. Implement updates and enhancements to address evolving regulatory requirements, technological advancements, and changing business needs.

III. MODELING AND ANALYSIS

This section shows how we implemented the desired properties of the smart contract as functions programmed in Solidity, a programming language for smart contracts deployed on the Ethereum network. Here we describe the essential code for the implementation of the proposed system. The proposed system relies on two smart contracts: the KYC smart contract and the token smart contract. For the token smart contract an open-source code based on ERC-20 recommendations is used (Buterin & Voglsteller, 2015). We made slight adjustments to

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:04/April-2024 Impact Factor- 7.868 www.irjmets.com

the ERC-20 code to simplify communications between the two smart contracts. The smart contract tracks the FIs addresses, the amount of tokens each FI transfers to the contract, and the hash imported initially or updated as a struct Bank

```
struct Bank {
  address bankAddress;
  bytes32 bankHash;
  uint256 balances;
}
address[] public banks_ids;
```

Figure 1: Structure of smart contract

The system records the FIs that have onboarded the customer in a list of addresses as banks_ids. banks_ids is what we have referred to as the list of onboarding institutions in previous sections. When a customer is added to the network the FI that carries out the initial KYC process creates the smart contract for that customer. The PoC uses a separate smart contract to deploy the smart contract used for KYC purposes. The deployment smart contract uses a function deploy Contract that takes as input the parameters necessary to deploy the KYC smart contract.

```
contract deployCheckHash {
  function deployContract(address newAddress, uint256 typeOf,
      address _bankAddress, string _hash, address tokenAddress)
      returns (address){
    return new checkHash(newAddress, typeOf, _bankAddress, _hash,
        tokenAddress);
  }
}
```

Figure 2: Deployment Smart Contract

The necessary inputs required by deployContract are the address of the customer (newAddress), the cost of the customer ($typeOf$), the address of the FI (_bankAddress), the hash of the KYC documents ($\_hash$), and the address of the token smart contract (tokenAddress). The token contract, as well as all the code, can be consulted in the GitHub.Repository "KYC-Optimized-and-Dynamic-System-using-Blockchain-Technology(Tth2549, 2017). The deployment smart contract uses its inputs to deploy the KYC smart contract as checkHash. The checkHash smart contract uses an initializing function to store the required information and transfer the ownership of the smart contract from the FI to the customer. The initializing function is only called when the smart contract is created. After ownership has been transferred the function calls storeProof to store the hash in the smart contract initializing function. Next, the FI is added to banks_ids and the hash is linked to the FI to record what hash the first FI imported. Further, the function uses the input typeOf to set the cost of the customer. Note that in this PoC we assume that there only exist three types of customers, which cost 100,000, 200,000, and 300,000 tokens, respectively. These values were chosen arbitrarily for illustration purposes. The customer is registered as the owner of the contract and therefore reserves the right to erase/kill the smart contract. To do this the customer needs to approach an FI that has access to the network. Ownership of the contract is then transferred from the customer to that FI so the function kill can be called and the contract erased.

After the KYC smart contract has been successfully deployed, it can be used by other FIs. The smart contract has a function, payment, that can be called by any FI. The payment function is used to inform FIs of the sum they must pay in order to interact with the smart contract.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:04/April-2024 Impact Factor- 7.868 www.irjmets.com

```
function checkHash(address newOwner, uint256 typeOf, address
    _bankAddress, string _hash, address tokenAddress) {
  transferOwnership(newOwner);
  banks[_bankAddress].bankAddress = _bankAddress;

  TokenAddress = tokenAddress;

  bytes32 proof = proofFor(_hash);
  storeProof(proof);
  banks[_bankAddress].bankHash = proof;
  banks_ids.push(_bankAddress);

  if (typeOf == 1) {
    contract_cost = 100000;
  }
  else if (typeOf == 2) {
    contract_cost = 200000;
  }
  else if (typeOf == 3) {
    contract_cost = 300000;
  }
  else throw;
}
```

Figure 3: Smart Contract initializing Function

```
function transferOwnership(address newOwner) onlyOwner {
  owner = newOwner;
}

function kill() {
  if (msg.sender == owner) {
  selfdestruct(owner);
  }
}
```

Figure 4: Smart Contract Function transfer Ownership

The payment function simply takes the cost of the customer and divides it by the number of FIs that have previously onboarded that customer plus one. If the FI accepts the amount to be paid, it can proceed to pay the given amount and use the smart contract further. The smart contract uses two functions to transfer the tokens—tokenFallback and payContract. tokenFallback stores the tokens in the smart contract and then uses payContract to distribute these tokens accordingly to other FIs

```
function payment() constant returns (uint256){
  return contract_cost/(banks_ids.length+1);
}
```

Figure 5: Smart Contract Function payment

Note that only after the tokenFallback function has verified that the amount transferred to the smart contract is

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:04/April-2024 Impact Factor- 7.868 www.irjmets.com

correct does it call payContract.

```
function tokenFallback(address from, uint256 amount, bytes data){
  if (amount == contract_cost/(banks_ids.length+1)){
    banks[from].balances += amount;
    banks[from].bankAddress = from;
    payContract();
    banks_ids.push(from);
  }
  else throw;
}
```

Figure 6: Smart Contract Function tokenFallback

The function pay Contract uses a simple loop to transfer the payment to all previous FIs. Note that the function token Fallback adds the new FI to the list of onboarding institutions, banks_ids, after the payContract function has been called. After the FI has paid the contract and the payment has been distributed, the FI can use the function check Document to compare the hash stored in the smart contract to the hash it has generated from the documents supplied by the customer.

```
function checkDocument(string document) alreadyPaid constant
    returns (string) {
  bytes32 proof = proofFor(document);
  return hasProof(proof);
}
```

Figure 7: Smart Contract Function check Document

The function check Document uses the modifier already Paid to ensure that the FI calling it is on the list of onboarding institutions—that is to say, that the FI has already paid its share.

```
modifier alreadyPaid {
  if (banks[msg.sender].bankAddress != msg.sender) throw;
}
```

Figure 8: Smart Contract Modifier alreadyPaid

If the FI has paid its share, checkDocument converts the input hash to the same format as the hash stored in the smart contract, using proof.

```
function proofFor(string document) constant returns (bytes32) {
  return sha256(document);
}
```

Figure 9: Smart Contract Function proofFor

Further, the function hasProof is used by the smart contract to compare the hash that results from hashing this document with the hash stored in the smart contract.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:04/April-2024 Impact Factor- 7.868 www.irjmets.com

```
function hasProof(bytes32 proof) constant returns (string) {
  if (proofs.length == 0) return "No data here.";
  if (proofs[proofs.length-1] == proof){
    return "Data is correct!";
  }
  else {
    for (uint256 i = 0; i < proofs.length; i++) {
      if (proofs[i] == proof) {
        return "Data is old, has been approved before." ;
      }
    }
  }
  return "Data has never been approved!";
}
```

Figure 10: Smart Contract Function hasProof

The has Proof function has three possible returns: “Data is correct!” if the hash is a match, “Data is old, has been approved before” if the hash that the FI is comparing has been updated by another FI and the new FI is using old documents, and “Data has never been approved!” if the hash does not match and has never been used before.

If the documents used for the KYC process need to be updated, the FI can update the hash in the smart contract using the function notarize.

```
function notarize(string document) alreadyPaid {
  bytes32 proof = proofFor(document);
  storeProof(proof);
}
```

Figure 11: Smart Contract Function notarize

Notarize converts the hash of the updated document to SHA256 and then adds the new, resulting hash to storage in the smart contract using store proof. The function notarize notes which FI updated the hash so there is a record of which FI has imported each specific hash. Additionally, the function keeps track of all the hashes used for the customer, in store proof.

```
function storeProof(bytes32 proof) internal {
  proofs.push(proof);
  if (banks_ids.length > 1){
    banks[msg.sender].bankHash = proof;
  }
}
```

Figure12: Smart Contract Function store Proof

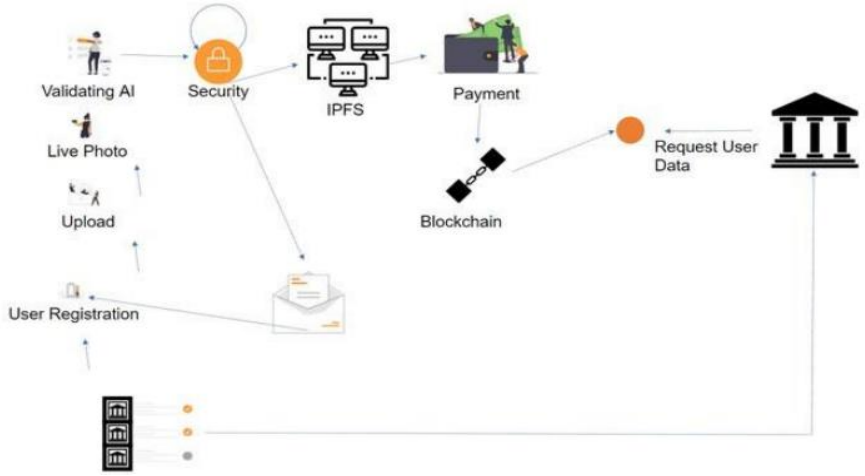

Figure 13: System architecture

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:04/April-2024 Impact Factor- 7.868 www.irjmets.com

Software requirements:

- Node Js
- Ganache
- Truffle
- React
- Meta mask

Hardware:

- Ram -4GB , Windows 10,11 ,Mac OS ,Linux
- Memory Size : 128gb

IV. RESULTS AND DISCUSSION

The project is implemented on a private Ethereum network. We used truffle version 5.0.2 to get a private Ethereum network and ganache to fund with ethers for usage in the project. The code is compiled with Solidity version 0.5.0.

Metamask is used as a bridge between the private Ethereum network created using truffle and the browser used (google chrome). The project ran on a local host using node version 16.13.1. All the transactions that are made are written in the Private Ethereum network with the account's id connected to the metamask. The gas fee will be detected from the same account with which the transactions are made.

Considering the high cost of Ethereum, the project is also implemented using a polygon. As polygon runs onlayer 2 Ethereum, we can run the project on the present private Ethereum network with just the change of currency used to pay the gas fee. Due to the polygon's low cost, the transactions' gas fees decreased significantly.

The project is implemented using Ethereum Blockchain provided by Ganache. A peer-to-peer connection is developed between users, and they will be able to either lend money or borrow money from others. The identities are verified in organisations using the KYC model proposed, and KYC verification is done. Every timea loan request is created or is responded to, it is added to the Blockchain as a transaction. The transactional fee while borrowing money is 0.00052697 ETH (Rs. 115.34), which is very high. And while investing money, the transactional fee is observed to be 0.00059638 ETH (.Rs. 130.54), which is high too. To decrease these additional charges while transactions, we used Polygon blockchain instead of Ethereum Blockchain as the gas fee for polygon blockchain is comparatively significantly less.

Table 1. Cost of transactions in the proposed lending model.

	Ethereum	Polygon
Need Money	Rs. 115.34 (0.00052697 ETH)	Rs. 0.07 (0.00069097 MATIC)
Invest Money	Rs. 130.54 (0.00059638 ETH)	Rs. 0.08 (0.00079573 MATIC)

Table 2. Cost of transactions in the proposed KYC model.

	Ethereum	Polygon
Adding KYC No. to Blockchain	Rs. 13.89 (0.00011746 ETH)	Rs. 0.01 (0.00016254 MATIC)
Requesting for authentication of KYC request	Rs. 24.92 (0.0002174 ETH)	Rs. 0.04 (0.0006054 MATIC)
Response for authentication of KYC request	Rs. 18.43 (0.00016494 ETH)	Rs. 0.02 (0.0003027 MATIC)

V. CONCLUSION

Any industry's future lies in complete digital transformation, which can only be achieved through infrastructure changes. To improve operational efficiency, core processes must be modified. This can only be accomplished by being open to new and disruptive technologies. The main goal of our proposed solution was to reimagine the

traditional KYC process. This proposed paper gives a solution to the problem of redundancy and inefficiency in the current KYC process, drastically lowering the system\'s operational costs. We also eliminate the presence of a single point of failure by utilising a blockchain-based approach. Blockchain is a game-changing technology, and its applications are expanding all the time. Implementing a blockchain application for kyc document verification provides proof of identity of a customer on bank and transparent access to all or any of the banks that are connected into the blockchain network, ensuring faster access to the kyc document while also providing security. By doing so, we can lower the cost of maintaining the document from the centralised organisation.

VI. REFERENCES

[1] Bharti Pralhad Rankhambe and Dr. Harmeet Kaur Khanuja, "Optimization of the KYC Process in the Banking Sector using Blockchain Technology",International Journal of Current Engineering and Technology, Special Issue-8 (Feb 2021)

[2] Ashok Kumar Yadav and Ramendra Kumar Bajpai, "KYC Optimization using Blockchain Smart Contract Technology",International Journal of Innovative Research in Applied Sciences and Engineering(IJIRASE) Volume 4, Issue 3, September 2020

[3] E. Sai Vikas Reddy, Nikhil Suhag and Manjunath S, "Know Your Customer (KYC) Process through Blockchain",International Research Journal of Engineering and Technology (IRJET),Volume: 07, Issue: 06 ,June 2020

[4] Syed Azhar Hussain and Zeeshan-ul-hassan Usmani, "Blockchain-based Decentralized KYC (Know-YourCustomer)",The Fourteenth International Conference on Systems and Networks Communications, ICSNC 2019

[5] Prof A. L. Maind, Pallavi Vijay Gedam, Snehal Kashinath Chavan and Chitrali Dnyaneshwar Shinde, "Kyc Using Blockchain",International Journal for Research in Engineering Application & Management (IJREAM), Special Issue – ICRTET-2018

[6] Abdullah Al Mamun, Sheikh Riad Has, Md Salahuddin Bhuiyan, M.Shamim Kaiser and Mohammad Abu Yousuf, "Secure and Transparent KYC for Banking System Using IPFS and Blockchain Technology", 2020 IEEE Region 10 Symposium

[7] Nikita Singhal, Mohit Kumar Sharma, Sandeep Singh Samant, Prajwal Goswami and Y.Abhilash Reddy, "Smart KYC Using Blockchain and IPFS", Springer Nature Singapore Pte Ltd. 2020

[8] Dr. Manoj Kumar, Nikhil, Parina Anand, "A Blockchain Based Approach For An Efficient Secure KYC Process With Data Sovereignty ",International Journal Of Scientific & Technology Research Volume 9, Issue 01, January 2020