

**PROOF OF KNOWLEDGE ON-CHAIN:  
A CREDENTIAL  
SYSTEM WITH MULTI-SIG**



**A PROJECT REPORT**

*Submitted by*

**JASON P  
(714221405002)**

*in partial fulfillment for the award of the degree  
of*

**MASTER OF ENGINEERING**

*in*

**COMPUTER SCIENCE AND ENGINEERING  
TAMILNADU COLLEGE OF ENGINEERING, COIMBATORE  
ANNA UNIVERSITY: CHENNAI 600 025**

**OCTOBER – 2023**

# **ANNA UNIVERSITY: CHENNAI 600 025**

## **BONAFIDE CERTIFICATE**

Certified that this Project titled **“PROOF OF KNOWLEDGE ON-CHAIN: A CREDENTIAL SYSTEM WITH MULTI-SIG”** is the bonafide work of **“JASON P (714221405002)”** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported here in does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

### **SIGNATURE**

**Dr. A. S. SHANTHI M.E., Ph.D.,  
HEAD OF THE DEPARTMENT,**

Department of Computer Science and  
Engineering,  
Tamilnadu College of Engineering,  
Coimbatore – 641 659.

### **SIGNATURE**

**Dr. A. S. SHANTHI M.E., Ph.D.,  
HEAD OF THE DEPARTMENT,  
SUPERVISOR,**

Department of Computer Science and  
Engineering,  
Tamilnadu College of Engineering,  
Coimbatore – 641 659.

**Submitted for the Anna University Viva-Voce held on .....**

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

## ACKNOWLEDGEMENT

First and foremost, I praise and thank parents and friends, from the bottom of our hearts for bestowing their benediction on us throughout the course of this project.

It is my pleasure to express my profound gratitude to the college management for admitting me into this project.

I extend my heartfelt and sincere thanks to our honorable principal, **Dr. M.KARTHIKEYAN B.E., M.Tech., Ph.D.**, for his support and encouragement in carrying out our project.

I am highly indebted and grateful to our beloved head of the department and project coordinator and guide **Dr. A.S.SHANTHI M.E., Ph.D.**, for her constant suggestion and persistent encouragement.

I also thank all the faculty members and lab in charge of our department who helped us to do this project successfully.

I wish to thank all teaching and non-teaching staff of our department, all hands and minds that helped me, all our friends and well-wishers who are all behind the success of this project.

## ABSTRACT

Verifying the originality of an academic transcript is a challenging issue in educational environment. This study addresses the problem of secure knowledge credential verification by introducing the Proof Of Knowledge On Chain (PoKoC) algorithm that can be used to prove the originality of an academic transcript. In many systems, verifying knowledge credentials has been a challenge, leading to potential fraud or errors. PoKoC solves this problem by extending the ERC721 standard and adding a Multi-Signature Scheme, significantly enhancing security. As a result, PoKoC enables the creation of tamper-proof public proofs that are exclusively owned by designated recipients. This research lays the foundation for innovative applications, emphasizing the crucial principles of immutability and transparency within decentralized ledger systems. By incorporating Zero-Knowledge Proofs (ZKPs) for private knowledge validation, PoKoC empowers the development of highly secure and private token-based systems, addressing the shortcomings of traditional knowledge verification. The elegant integration of cryptographic techniques and decentralized principles within the PoKoC algorithm holds significant promise for the future of secure digital knowledge management, promising improved trust and accuracy in various fields. The analysis of PoKoC's security audit scores further reaffirms its robustness, surpassing other models, and underlining its potential for ensuring trustworthy knowledge verification.

# TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	iv
	TABLE OF CONTENTS	v
	LIST OF TABLES	viii
	LIST OF FIGURES	ix
	LIST OF TERMINOLOGIES	x
<b>CHAPTER 1</b>	<b>INTRODUCTION</b>	<b>2</b>
1.1	Blockchain	3
1.1.1	Layer 1 Blockchain	4
1.1.2	Layer 2 Blockchain	4
1.1.3	On-chain Data	6
1.2	Zero Knowledge Proofs (ZKPs)	7
1.3	ERC TOKEN	8
1.4	Wallet & Signatures	9
1.5	Multi-Sig:	9
1.6	Potential Applications of PoKoC	10
<b>CHAPTER 2</b>	<b>LITERATURE SURVEY</b>	<b>15</b>
2.1	"Bitcoin: A Peer-to-Peer Electronic Cash System"	15
2.2	"Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform"	15
2.3	"Polygon Litepaper"	16
2.4	"Hyperledger Foundation"	17
2.5	"Use of Elliptic Curves in Cryptography"	17
2.6	" A Digital Signature Based on A Conventional Public Key Cryptosystem"	18
2.7	"The secure hash algorithm-2 (SHA-2) family of cryptographic hash functions"	18
2.8	"On The Size of Pairing-Based Proofs "	19

2.9	Decentralized Society: Finding Web3's Soul.	19
2.10	"EIP721: ERC-721 Non-Fungible Token Standard."	20
2.11	Blockchain in education management: present and future applications	20
2.12	COMPARATIVE CASE STUDY	21
<b>CHAPTER 3 EXISTING SYSTEM</b>		<b>24</b>
3.1	Password based Credential Authentication Systems	24
3.2	OpenID Connect (OIDC) Protocol[25]	24
3.3	Security Assertion Markup Language (SAML) Protocol[26]	25
3.4	Drawbacks of Existing Systems	26
<b>CHAPTER 4 PROPOSED SYSTEM</b>		<b>28</b>
4.1	Algorithm	28
4.1.1	Motivation	29
4.2	The Overall Design of the Proposed System.	29
4.3	Methodology	30
4.4	Non-Transferable Token	30
4.5	Multi-Signature Scheme	31
4.6	Proof Of Knowledge	31
4.7	Public Credentials	32
4.8	Private Credentials & Hashed Proofs	32
4.9	Architectural View	33
4.10	Advantages Of PoKoC	33
4.11	Summary	34
<b>CHAPTER 5 SYSTEM SPECIFICATION</b>		<b>36</b>
<b>CHAPTER 6 SOFTWARE TECHNOLOGIES USED</b>		<b>38</b>
6.1	Solidity	38
6.2	ZK-Snark	38
6.3	Zokrates	39

<b>6.4</b>	<b>EthersJS</b>	<b>39</b>
<b>6.5</b>	<b>SolidityScan</b>	<b>40</b>
<b>CHAPTER 7 DESIGN OF POKOC</b>		<b>42</b>
<b>7.1</b>	<b>Graphical Design of PoKoC</b>	<b>43</b>
<b>7.2</b>	<b>Verification of PoKoC</b>	<b>44</b>
<b>CHAPTER 8 RESULTS AND DISCUSSION</b>		<b>47</b>
<b>8.1</b>	<b>Metrics Used for Evaluation</b>	<b>47</b>
<b>8.2</b>	<b>Security Levels</b>	<b>47</b>
<b>8.3</b>	<b>Overall Security Score Calculation</b>	<b>48</b>
<b>8.4</b>	<b>5.2 Tables and Graphs Analysis</b>	<b>48</b>
<b>CHAPTER 9 CONCLUSION AND FUTURE WORK</b>		<b>54</b>
<b>APPENDIX – 1 SAMPLE CODE</b>		<b>57</b>
<b>APPENDIX – 2 SCREENSHOTS</b>		<b>85</b>
<b>REFERENCES</b>		<b>92</b>

## LIST OF TABLES

TABLE NO.	TITLE	PAGE NO.
Table 1.	Comparative Case Study .....	21
Table 2.	Security Audit Scores.....	48
Table 3.	System Scores for Different Configurations .....	51



## LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
Figure 1 :	The Overall Architecture of the Proposed System .....	30
Figure 2 :	Architecture of Non-Transferable Tokenomics .....	33
Figure 3 :	Directed Acyclic Property of PoKoC .....	43
Figure 4 :	Design Flow of PoKoC .....	44
Figure 5 :	Security Score Comparison .....	49
Figure 6 :	Single Contract and Overall System.....	50
Figure 7 :	Security Score of Single Contract .....	51
Figure 8 :	Security Score of Overall System.....	51

## LIST OF TERMINOLOGIES

**Blockchain:** A distributed digital ledger that records transactions on a network of computers.

**Layer 1 Blockchain (Ethereum):** A blockchain that runs its own network and allows for the creation of smart contracts and decentralized applications (dApps).

**Layer 2 Blockchain (Polygon):** A blockchain that is built on top of an existing blockchain (in this case, Ethereum) to improve scalability and reduce costs.

**On-chain data:** Data that is stored directly on a blockchain.

**Single-Signature:** A type of digital signature that requires only one key to sign and authorize a transaction.

**Multiple-Signature:** A type of digital signature that requires more than one key to sign and authorize a transaction.

**Zero Knowledge Proofs:** A cryptographic method for proving the authenticity of information without revealing any additional information.

**Proof of Stake:** A consensus mechanism used in blockchain networks where validators are chosen based on the amount of cryptocurrency they hold.

**Zero Knowledge Witness:** A type of ZKP that allows a user to prove knowledge of a secret without revealing the secret itself.

**Succinct Proof:** A type of proof that can be verified in a short amount of time and with limited resources.

**Merkle Proofs:** A cryptographic technique for proving the membership of an element in a set.

**Non-Reusable Proof:** A type of proof that can only be used once and cannot be reused.

**Private Proof:** A type of proof that is kept confidential and only shared with specific parties.

**Public Proof:** A type of proof that is publicly available and can be verified by anyone.

**Issuer:** A party that creates and issues credentials or tokens.

**Validator:** A party that verifies the authenticity of credentials or tokens.

**Credentials Aggregator:** A party that collects and verifies credentials from multiple issuers.

**Block Time:** The time it takes for a new block to be added to the blockchain.

**ZK SNARK:** A type of ZKP that allows for efficient verification and is commonly used in blockchain networks.

**Token:** A digital asset that represents a unit of value or ownership.

**Non-Transferable Tokenomics:** A set of rules that govern the issuance and ownership of tokens that cannot be transferred.

**Open Ledger:** A public blockchain ledger that is transparent and visible to anyone.

Smart Contracts: Self-executing contracts that automatically enforce the rules and regulations of an agreement.

Smart Contract Wallet: A digital wallet that is capable of executing smart contracts.

Infeasible: A term used to describe a task that is computationally impossible to complete within a reasonable amount of time.

Immutable: A term used to describe data that cannot be changed once it has been added to the blockchain.

Decentralized: A term used to describe a system that operates without a central authority or intermediary.

dApp: A decentralized application that runs on a blockchain network.

# **CHAPTER 1**

# CHAPTER 1

## INTRODUCTION

In the knowledge and educational domains, verifying educational and academic credentials has become a critical concern. This issue has gained prominence not only due to the recent Accenture[23], which highlighted the problem, but also because of numerous undisclosed education-related scams. These incidents underscore the substantial challenges organizations face in ensuring the authenticity of documents and experience letters provided by job applicants.

To effectively tackle these challenges, this study introduces the Proof Of Knowledge On Chain (PoKoC) algorithm, primarily designed to create Non-Transferable Tokens (NTTs) for knowledge credentials. PoKoC extends the ERC721[11] standard and integrates a Multi-Signature Scheme, significantly enhancing security and ensuring the tamper-proof nature of the credentials. To enhance verification and privacy, we also integrate zero-knowledge proofs. This research serves as the foundational step towards pioneering applications that place a strong emphasis on the principles of immutability and transparency within decentralized ledger systems, making a decentralized framework available for various knowledge and educational domains.

In this project, we propose a novel system for proving knowledge on a blockchain, utilizing smart contracts. The proposed system is developed using three major abstracted smart contracts, 1. NTT protocol, 2. Multi-Signature (Multi-Sig), with Zero-Knowledge-Proof(ZKP) protocol, which allows signing entities to issue credentials to specific users on the blockchain, without the risk of fraud or intermediary manipulation. We argue that our approach offers significant advantages over existing solutions, including greater immutability, security, transparency, and flexibility, as well as lower costs and more efficient verification.

The scope of this project is to implement an immutable credential system with multi-signature schemes, In which educational systems(remote/physical) can provide non-transferable credentials to students. Usually these type of tokens have no monetary value but it holds tamper-proof credentials .

The implemented smart contracts will be audited and tested with the standard blockchain security practices. The deployment choice of the L2 blockchain makes this implementation low cost and sustainable for the environment. To support our claims, we provide a formal analysis of the security properties of our system and demonstrate its practical feasibility through a proof-of-concept implementation.

## **1.1 Blockchain**

Blockchain is a decentralized and distributed digital ledger technology that allows for the secure and transparent recording of transactions. It is best known as the underlying technology behind cryptocurrencies such as Bitcoin, but it has many other potential applications in a wide range of industries, from finance to healthcare to supply chain management.

At its core, a blockchain is a database that is maintained by a network of nodes or computers, rather than a central authority. Each node on the network has a copy of the database, and every transaction that occurs on the network is verified and recorded by multiple nodes in a secure and tamper-proof way. This makes it very difficult for anyone to modify or manipulate the data on the blockchain without being detected.

One of the key features of blockchain technology is its use of cryptographic algorithms to ensure the integrity and security of the data on the network. Transactions are secured using complex mathematical algorithms that make it virtually impossible for anyone to tamper with the data without being detected.

Another important feature of blockchain technology is its transparency. Because every transaction on the blockchain is recorded and verified by multiple nodes, it is possible to trace the history of a particular asset or transaction all the way back to its origin. This makes it a powerful tool for ensuring transparency and accountability in a wide range of industries.

Overall, blockchain technology has the potential to revolutionize the way we conduct business and manage data in a wide range of industries. Its decentralized and secure nature makes it an attractive option for organizations looking to improve efficiency, security, and transparency.

### **1.1.1 Layer 1 Blockchain**

Layer 1 blockchain refers to the underlying blockchain technology that forms the foundation of a decentralized network. This layer includes the core protocol and consensus mechanism that define the rules and processes for validating and adding new transactions to the blockchain.

Bitcoin, the first and most well-known blockchain platform, is an example of a layer 1 blockchain. It uses a proof-of-work (PoW) consensus mechanism to validate transactions and add new blocks to the blockchain[1][2]. This means that miners compete to solve complex mathematical puzzles, and the first one to solve the puzzle gets to add the next block of transactions to the blockchain.

Another example of a layer 1 blockchain is Litecoin, which was designed to be a faster and more lightweight version of Bitcoin. It uses a similar PoW consensus mechanism, but with a different hashing algorithm that is less computationally intensive.

Ethereum is another example of a layer 1 blockchain, but it is designed to be more than just a cryptocurrency. It allows developers to build and deploy decentralized applications (DApps) and smart contracts on top of the blockchain, using its own programming language called Solidity. Ethereum also uses a PoW consensus mechanism, but it is currently in the process of transitioning to a proof-of-stake (PoS) consensus mechanism, which is designed to be more energy-efficient and scalable.

Overall, layer 1 blockchains form the foundation of the decentralized web, providing the infrastructure and rules for validating and adding new transactions to the blockchain. While there are many layer 1 blockchains in existence, each with their own unique features and strengths, they all share the goal of enabling decentralized and trustless systems that can operate without a central authority.

### **1.1.2 Layer 2 Blockchain**

Layer 2 blockchain refers to a secondary layer of technology that is built on top of a layer 1 blockchain, such as Ethereum or Bitcoin. The purpose of a layer 2 blockchain is to improve the scalability, security, and functionality of the underlying layer 1 blockchain.

Layer 2 solutions can take many different forms, but some common examples include:

**Payment channels:** These are off-chain channels that allow two parties to transact with each other without having to record every transaction on the layer 1 blockchain. This can significantly improve the speed and cost-effectiveness of transactions, especially for small or frequent payments.

**State channels:** These are similar to payment channels, but they allow for more complex interactions between multiple parties, such as playing games or executing smart contracts. State channels can greatly reduce the cost and complexity of executing complex transactions on the layer 1 blockchain.

**Sidechains:** These are separate blockchains that are interoperable with the main layer 1 blockchain. Sidechains can allow for faster and more efficient transactions, as well as support for additional features and functionality that may not be available on the main layer 1 blockchain.

**Plasma:** This is a scaling solution developed for Ethereum that involves creating a network of interconnected sidechains that can handle large volumes of transactions. Plasma is designed to be highly scalable, allowing for millions of transactions per second while still maintaining the security and decentralization of the underlying layer 1 blockchain.

One of the key advantages of layer 2 blockchains is that they allow for significant improvements in scalability without compromising on the security and decentralization of the underlying layer 1 blockchain. They also provide developers with greater flexibility and freedom to build complex applications and use cases on top of the blockchain.

However, layer 2 solutions can also introduce additional complexity and technical challenges, such as the need for strong interoperability and security standards between the different layers. Overall, layer 2 blockchain technology represents an important area of innovation and development in the blockchain space, as developers seek to create more scalable and functional blockchain applications that can support a wide range of use cases.

Polygon[6] (formerly Matic Network) is a Layer 2 scaling solution for Ethereum that aims to improve scalability and usability for decentralized applications (DApps) and the broader blockchain ecosystem. Layer 2 solutions are designed to improve the scalability of



existing blockchain platforms by moving some of the processing and transaction load off of the main chain and onto secondary layers.

The Polygon network uses a combination of sidechains, plasma chains, and state channels to provide faster and cheaper transactions while still maintaining the security and decentralization of the Ethereum network. By using these Layer 2 technologies, Polygon is able to process up to 65,000 transactions per second (TPS), compared to Ethereum's current limit of around 15 TPS.

One of the key benefits of Polygon is its compatibility with existing Ethereum infrastructure, including smart contracts and decentralized applications. This makes it easy for developers to migrate their existing DApps and projects to the Polygon network, without having to make major changes to their code.

Another advantage of Polygon is its low transaction fees. Because transactions are processed on Layer 2, users can avoid the high fees and congestion that are common on the Ethereum main chain. This makes it more accessible and affordable for users to participate in the Polygon ecosystem.

Overall, Polygon is a promising Layer 2 solution that has the potential to improve the scalability and usability of the Ethereum network. By providing faster, cheaper, and more efficient transactions, Polygon is helping to drive the adoption and growth of decentralized applications and the broader blockchain ecosystem.

### **1.1.3 On-chain Data**

On-chain data refers to any data that is stored directly on a blockchain, such as Bitcoin or Ethereum. This can include transaction data, account balances, smart contract code, and other information that is recorded and stored on the blockchain.

One of the key advantages of storing data on a blockchain is that it is decentralized and immutable, meaning that once data is recorded on the blockchain, it cannot be changed or altered in any way. This makes blockchains a highly secure and trustworthy way to store sensitive information, such as financial transactions or identity information.

On-chain data is also accessible to anyone with a copy of the blockchain, which means that it can be easily audited and verified by anyone who wants to review it. This makes

blockchains a transparent and open platform for storing and sharing information, which is particularly important in areas such as finance or supply chain management.

However, storing large amounts of data directly on a blockchain can also be resource-intensive and can slow down the overall performance of the network. This has led to the development of various off-chain data storage solutions, such as IPFS or Swarm, that allow developers to store data in a more decentralized and scalable way while still leveraging the security and trustlessness of the underlying blockchain.

Overall, on-chain data represents an important aspect of the blockchain ecosystem, providing a secure and transparent way to store and share information. As blockchain technology continues to evolve and mature, it is likely that we will see new and innovative use cases for on-chain data emerge, particularly in areas such as decentralized finance, identity management, and supply chain management.

## 1.2 Zero Knowledge Proofs (ZKPs)

ZKPs are a type of cryptographic protocol that allows one party to prove to another party that a given statement is true, without revealing any additional information beyond the statement itself. ZKPs are used to prove knowledge of information or to authenticate a user or device, without the need for the user or device to reveal any sensitive information.

In a ZKP, the party making the statement (known as the prover) generates a proof that the statement is true, while the party receiving the proof (known as the verifier) can verify that the proof is valid without learning any additional information beyond the truth of the statement. ZKPs have a wide range of potential applications, including:

**Privacy-preserving authentication:** ZKPs can be used to authenticate a user or device without revealing any sensitive information about the user or device, such as a password or private key.

**Privacy-preserving transactions:** ZKPs can be used to prove that a transaction is valid without revealing the specific details of the transaction, such as the sender or recipient addresses.

**Secure voting:** ZKPs can be used to ensure that a vote is valid and counted without revealing the identity of the voter or the specific details of their vote.

**Secure data sharing:** ZKPs can be used to share data between parties in a secure and verifiable way, without revealing any sensitive information beyond the specific data being shared.

ZKPs are an active area of research in the field of cryptography, with ongoing efforts to improve their efficiency and scalability for use in a wide range of applications. As blockchain technology continues to evolve and mature, it is likely that we will see increasing use of ZKPs in various blockchain-based applications, particularly in areas such as privacy, security, and authentication.

### **1.3 ERC TOKEN**

ERC (Ethereum Request for Comment) tokens are a type of digital asset that is created and hosted on the Ethereum blockchain network. These tokens comply with a set of standardized rules and guidelines known as ERC standards, which define the functionality and behaviors of the token. Each with its own set of functionalities and features. Here we are going to focus on the credential system related tokens

#### **ERC-20:**

This is the most widely used token standard on the Ethereum network. ERC-20[9] tokens are fungible, meaning they are interchangeable with each other and have a fixed supply. They are used to represent assets such as currencies, commodities, and securities.

#### **ERC-721:**

This standard defines a non-fungible token (NFT) that is unique and indivisible. Each ERC-721 token is one-of-a-kind, and it can represent digital assets such as artwork, collectibles, and game items.

#### **ERC-1155:**

This standard defines a multi-token contract that allows for the creation of both fungible and non-fungible tokens within the same contract. ERC-1155[27] tokens are often used for in-game items, where both unique and interchangeable items are needed.

In this paper the smart contracts for the immutable credential system is forked from the ERC721. It is rebuild with non transferability functionality.

## **1.4 Wallet & Signatures**

In Ethereum[6], a wallet is a software program that allows users to securely store, manage, and transfer their digital assets, including Ether (ETH) and other ERC tokens. There are different types of wallets, including desktop wallets, mobile wallets, hardware wallets, and web wallets. Each type of wallet has its unique features, advantages, and disadvantages, and users should choose the one that suits their needs and preferences. One of the essential features of Ethereum wallets is their ability to handle cryptographic signatures. When a user initiates a transaction, the wallet creates a digital signature that proves the ownership and authenticity of the transaction. This signature is then broadcasted to the Ethereum network for validation and processing by the miners.

Ethereum uses a public-key cryptography algorithm called Elliptic Curve Digital Signature Algorithm (ECDSA) to generate and verify signatures. Each Ethereum account has a public key and a private key, which are mathematically related. The private key is used to create the digital signature, while the public key is used to verify it. The private key should always be kept secure and never shared with anyone, as it grants access to the user's digital assets.

In Ethereum, multi-signature wallets are also available, which require multiple signatures from authorized parties before a transaction can be executed. This adds an extra layer of security and ensures that no single party can unilaterally execute transactions. Multi-signature wallets are commonly used by organizations, businesses, or groups of individuals that require shared control over a wallet's assets.

## **1.5 Multi-Sig:**

Multi-sig is a term used to describe a cryptographic technique used in blockchain technology that requires multiple parties to sign a transaction before it can be executed. In a traditional transaction, a single private key is used to sign the transaction and authorize the transfer of funds or assets. However, in a multi-sig transaction, multiple private keys are required to sign the transaction before it can be executed.

The number of required signatures is determined by a threshold value, which can be set to any number. For example, a 2-of-3 multi-sig transaction requires at least two of the three private keys to sign the transaction. This ensures that no single party has complete control over the funds or assets being transferred, and increases the security and integrity of the transaction.

Multi-sig is achieved through the use of public-private key pairs. Each party involved in the multi-sig transaction has their own private key, which is used to sign the transaction. The public key associated with each private key is recorded on the blockchain, allowing anyone to verify the signatures and ensure that the transaction is legitimate. Multi-sig is widely used in blockchain applications to increase the security and reliability of transactions. It is particularly useful in situations where a single point of failure could lead to catastrophic consequences, such as the transfer of large sums of money or the management of critical infrastructure. Multi-sig can be used to establish hierarchical authorities in a credential system. In our implementation, in a university setting, a student's knowledge and skills may be assessed by multiple authorities, such as their instructor, department head, and dean. Each authority could have a designated key or set of keys that must sign off on the issuance of a credential. In this way, the multi-sig system can ensure that the issuance of credentials is authorized by all relevant parties, while also providing transparency and accountability. Additionally, the use of an immutable credential system can prevent fraudulent issuance or misuse of credentials by unauthorized individuals.

## **1.6 Potential Applications of PoKoC**

The Proof of Knowledge onChain has potential applications in various knowledge management sectors:

### **Education and Certification:**

In the dynamic world of education, credential verification remains a challenge. PoKoC promises a transformative solution. Imagine educational institutions leveraging this technology to create Non-Transferable Tokens (NTTs) representing degrees and certifications. These NTTs would serve as secure, blockchain-backed digital credentials, bolstering the authenticity and credibility of academic achievements. With public proofs stored on-chain to ensure transparency and private Zero-Knowledge Proofs (ZKPs) guarding sensitive academic data, this innovation has the potential to reshape the educational landscape.

**Professional Licensing and Accreditation:**

Verifying professional qualifications is often a tedious process. PoKoC offers a tantalizing prospect - regulatory bodies issuing NTTs on the blockchain. These tokens would provide an unalterable record of an individual's qualifications, fostering trust in the services professionals offer. Public proofs on-chain would instill confidence, while private ZKPs would secure sensitive professional data, promising a robust solution.

**Identity Verification:**

Online identity verification is riddled with challenges, including the risk of identity fraud. Enter PoKoC. Picture a world where individuals possess NTTs containing verified personal information, strengthening online identity verification processes. Public proofs would ensure trust in verified identities, while private ZKPs would act as guardians of sensitive personal data, enhancing privacy and security.

**Supply Chain and Product Authenticity:**

The battle against counterfeit products is a constant struggle for businesses across industries. PoKoC can turn the tide by enabling businesses to demonstrate product authenticity on the blockchain. With public proofs stored on-chain, consumers could easily verify product origins and authenticity. Meanwhile, private ZKPs would stand as sentinels guarding sensitive supply chain data, putting an end to data leaks.

**Research and Scientific Credentials:**

The world of research and academia can benefit greatly from PoKoC. Imagine researchers and scientists using it to validate their qualifications and contributions within their respective fields, fostering transparency and trust. Public proofs on-chain would offer a transparent view of research credentials, while private ZKPs would protect sensitive research data, preserving intellectual property.

**Government and Public Records:**

The management of vital public records is often marred by concerns of tampering and administrative inefficiencies. PoKoC presents an enticing solution. Governments issuing blockchain-backed NTTs for vital documents would ensure the integrity and transparency of

public records. Public proofs stored on-chain would guarantee transparency, while private ZKPs would fortify sensitive data, promising a streamlined and secure system.

### **Healthcare and Medical Credentials:**

In healthcare, patient safety hinges on the qualifications of professionals. PoKoC could revolutionize this aspect by validating healthcare qualifications on the blockchain. Public proofs on-chain would make medical credential verification a breeze, while private ZKPs would ensure the confidentiality of sensitive healthcare data.

### **Legal Contracts and Agreements:**

Legal transactions are often bogged down by paperwork and disputes. PoKoC can simplify this landscape by using NTTs to represent legally binding agreements on the blockchain. Public proofs on-chain would ensure transparency in legal agreements, while private ZKPs would act as guardians of confidential legal data, facilitating dispute resolution.

### **Privacy-Preserving Systems:**

Privacy is paramount in various systems, and PoKoC, with its incorporation of ZKPs, paves the way for privacy-preserving solutions. Picture secure voting systems, authentication methods, and access controls where knowledge verification is necessary without revealing sensitive information. Public proofs on-chain would ensure transparency in system operations, while private ZKPs would safeguard sensitive user data, promising both privacy and security.

### **Decentralized Finance (DeFi):**

Security is at the heart of DeFi, and PoKoC aims to enhance it. With NTTs as proof of knowledge, public proofs on-chain would guarantee transparency in financial credentials, while private ZKPs would secure financial privacy, protecting sensitive financial data in the realm of decentralized finance.

### **Human Resources and Job Credentials:**

Streamlining employee qualification verification is a pressing need. PoKoC could revolutionize HR departments by allowing them to issue NTTs for employee qualifications, simplifying the verification process for job applicants and employers. Public proofs on-chain

would ensure transparency in job credentials, while private ZKPs would safeguard HR data, maintaining confidentiality.

### **Intellectual Property and Copyright:**

The protection of intellectual property is paramount in our digital age. PoKoC aids in establishing proof of ownership and authorship for intellectual property. Public proofs on-chain would enable easy verification, while private ZKPs would secure sensitive IP data, preserving the rights of creators and innovators.

### **Cybersecurity:**

In the ever-evolving realm of cybersecurity, expertise validation is critical. PoKoC steps in to validate the knowledge and certifications of cybersecurity professionals. Public proofs in cybersecurity credentials stored on-chain would instill trust, while private ZKPs would secure sensitive cybersecurity data, ensuring the confidentiality of critical information.

The PoKoC algorithm, with its integration of on-chain public proofs and private ZKPs, holds the potential to revolutionize knowledge verification across diverse applications. It promises to enhance security, privacy, and transparency in digital knowledge management, marking a pivotal shift that awaits implementation.



## **CHAPTER 2**

## **CHAPTER 2**

### **LITERATURE SURVEY**

Proof of Knowledge On-Chain (PoKoC) is a relatively new concept in the blockchain and zero-knowledge proof (ZKP) space, and as such, there are only a few publications that directly address the topic. However, there are some papers that discuss the underlying technology and concepts that are relevant to PoKoC. The following are some of the notable previous works related to the area.

#### **2.1 Bitcoin: A Peer-to-Peer Electronic Cash System**

**Authors: Satoshi Nakamoto.**

This paper stands as the genesis of an actual electronic cash system intricately woven into an innovative blockchain architecture, consisting of interlinked blocks, meticulously outlined by Satoshi Nakamoto in 2008. This pioneering work not only introduced the concept of Bitcoin but also served as the foundational platform for contemporary blockchain technology. At its heart, Bitcoin embodies a peer-to-peer timestamp server-client mechanism that seamlessly integrates transparency with unwavering security. This intricate system operates through the astute utilization of proof-of-work (PoW), while being fortified by the formidable cryptographic techniques, including Merkle trees, Elliptic Curve Cryptography (ECC), and the Secure Hash Algorithm (SHA). Bitcoin, in its essence, transcends the realm of digital currency; it signifies an epochal leap in the evolution of financial technology. Beyond its profound technical intricacies, this paper serves as the bedrock upon which subsequent innovations in the blockchain ecosystem have flourished, leaving an indelible mark on the landscape of finance and technology.

#### **2.2 Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform**

**Authors: Vitalik Buterin, Gavin Wood, Jeffrey Wilcke, Anthony Di Iorio, Charles Hoskinson.**

In the realm of blockchain innovation, the Ethereum whitepaper, published in 2013, marks a monumental milestone. Ethereum, introduced in 2013 and officially launched in 2015, redefined the possibilities of blockchain technology. It operates as a decentralized blockchain and introduced Ether (ETH) as its native cryptocurrency, but its significance

extends far beyond mere digital currency. Ethereum serves as the bedrock for decentralized applications (DApps), including transformative innovations like Decentralized Finance (DeFi), Non-Fungible Tokens (NFTs), and ERC-20 tokens, each of which has left an indelible mark on the blockchain landscape. A pivotal moment in Ethereum's journey came in 2022 when it transitioned to a proof-of-stake (PoS) consensus mechanism, reducing energy consumption by a staggering 99%. The core innovation of Ethereum lies in its Ethereum Virtual Machine (EVM) and programmable smart contracts, enabling the creation of powerful, self-executing agreements and a flourishing ecosystem of DApps. Of particular note is Ethereum Improvement Proposal (EIP) 721, commonly known as the ERC721 standard, which introduced the concept of non-fungible tokens (NFTs), fundamentally revolutionizing the representation and ownership of unique assets on the blockchain. This Ethereum whitepaper, an authoritative source of Ethereum's foundational principles, can be accessed online. Its pages offer an essential reference for comprehending the profound impact of Ethereum on blockchain technology and its applications.

## **2.3 Polygon Litepaper**

**Authors: Jaynti Kanani, Sandeep Nailwal, Anurag Arjun.**

In the context of blockchain innovation, the 'Polygon Litepaper' authored by Jaynti Kanani, Sandeep Nailwal, and Anurag Arjun represents a pivotal resource. Formerly known as the Matic Network, Polygon Layer-2 networks offer solutions to enhance transaction speed and cost-effectiveness within the Ethereum blockchain, directly addressing scalability issues (Kanani et al., 2022). This enlightening document identifies the challenges of current blockchains, including slow transaction times and high fees, proposing a remedy through sidechains that achieve scalability while being secured by a robust Proof-of-Stake mechanism. Notably, Polygon sidechains seamlessly integrate with the Ethereum Virtual Machine (EVM), allowing the effortless migration of existing Ethereum applications, making it an attractive choice for developers seeking to build scalable decentralized applications. Moreover, the 'Polygon Litepaper' outlines the benefits of adopting Polygon for enterprises, including reduced transaction costs and improved transaction throughput, making it an enticing proposition for businesses considering blockchain technology integration. Overall, this document serves as an invaluable resource for anyone interested in exploring Polygon's capabilities or contemplating its utilization in their projects.

## **2.4 Hyperledger Foundation**

**Authors: Hyperledger Technical Steering Committee (TSC) and the Hyperledger Architecture Working Group (AWG).**

In the landscape of blockchain technology, the establishment of Hyperledger by the Linux Foundation in 2015 marks a pivotal milestone, offering an extensive suite of open-source, enterprise-grade blockchain frameworks, tools, and libraries governed by a diverse community of stakeholders ("Linux Foundation Unites Industry Leaders to Advance Blockchain Technology - The Linux Foundation"). Hyperledger's notable strengths lie in its focus on privacy and permissioning, making it an ideal choice for businesses seeking to safeguard sensitive data and control blockchain access. Furthermore, its modularity and flexibility enable tailored solutions for various industries, including finance, healthcare, and supply chain management. The practical adoption of Hyperledger is evidenced by its use in innovative applications by businesses such as Walmart for supply chain tracking, JPMorgan Chase for blockchain-based payments, and IBM for medical record management, demonstrating its significance in driving blockchain innovation across sectors.

## **2.5 Use of Elliptic Curves in Cryptography**

**Authors: Miller, V.**

Elliptic Curve Cryptography (ECC) is a cryptographic technique that relies on the mathematics of elliptic curves over finite fields. This innovative approach was first introduced and detailed by Victor Miller in his pivotal paper titled "Use of Elliptic Curves in Cryptography" in 1986, presented in the proceedings of Advances in Cryptology — CRYPTO '85. In this landmark work, Miller proposed an analogue of the Diffie-Hellman key exchange protocol using elliptic curves, demonstrating its superiority over the traditional method based on finite fields ( $GF(p)$ ) in terms of both speed and security.

The significance of ECC has grown over the years, particularly as computational power has continued to advance. Its efficient algorithms and strong security properties have made it a cornerstone of modern cryptography. ECC's application extends far beyond traditional cryptographic practices; it plays a pivotal role in securing key management within blockchain networks, where the need for robust and efficient cryptographic techniques is paramount. In essence, ECC's elegance lies in its ability to provide a high level of security with relatively small key sizes, making it an essential tool in ensuring the integrity and confidentiality of data in various digital contexts. Miller's pioneering work paved the way for

the widespread adoption of ECC, making it an integral part of contemporary cryptographic practices, especially within the dynamic and evolving landscape of blockchain technology.

## **2.6 A Digital Signature Based on A Conventional Public Key Cryptosystem**

**Authors: Merkle, R. C.**

The Merkle tree, as described by Ralph Merkle in his seminal work in 1988, is a hierarchical data structure where each "leaf" node is associated with the cryptographic hash of a specific data block. However, nodes that are not leaves, often referred to as branches, inner nodes, or inodes, are labeled with the cryptographic hash of their child nodes' labels. This ingenious design forms the basis of the Merkle tree's efficiency and security.

In practical terms, a Merkle tree allows for the rapid and reliable verification of the contents of a large data structure. It accomplishes this by systematically aggregating data into a hierarchical structure of hashes, ultimately resulting in a single hash value at the tree's root. This root hash serves as a concise representation of all the data within the tree, making it highly efficient for integrity checks. The Merkle tree concept is an extension of both hash lists and hash chains, offering a more versatile and structured approach to data verification. In the context of blockchain technology, Merkle trees play a pivotal role in ensuring data integrity and operational efficiency. Within a blockchain, the transactions of each block are organized into a Merkle tree, providing a secure and compact means of verifying the contents of a given block. This not only enhances data security but also optimizes database operations, making the Merkle tree a fundamental component of blockchain technology.

## **2.7 The Secure Hash Algorithm-2 (SHA-2) Family of Cryptographic Hash Functions**

**Authors: Penard, S., & van Werkhoven, L.**

The Secure Hash Algorithm 2 (SHA-2) family of cryptographic hash functions, as explored by Penard and van Werkhoven in 2016, represents a critical milestone in the realm of cryptographic security. It was designed by the National Security Agency (NSA) in 2001 and was subsequently approved, as noted in the Federal Register Notice 02-21599 in 2002. Within the blockchain ecosystem, the SHA-2 family, particularly SHA-256, plays a pivotal role in ensuring data integrity, a cornerstone of blockchain technology. SHA-256, as part of the SHA-2 family, is instrumental in rendering blockchain transactions tamper-proof and immutable. Its mechanism involves generating a unique cryptographic hash for each block of

data, creating an interlinked and secure chain of blocks. Miners within the blockchain network employ SHA-256 as a fundamental component of the proof-of-work consensus mechanism, using it to validate and add new blocks to the blockchain. This robust and proven cryptographic function underpins the trustworthiness of blockchain networks, making it an integral element in the world of digital ledger technology.

## **2.8 On The Size of Pairing-Based Proofs**

**Authors: Groth, J.**

In the realm of modern cryptography, Groth's pioneering work in 2016, titled "On the size of pairing-based proofs," stands as a seminal contribution that has significantly influenced the landscape of zero-knowledge proofs (ZKPs). This cryptographic innovation empowers one party, known as the prover, to establish the veracity of a statement to another party, the verifier, without revealing any extraneous information. This non-interactive protocol efficiently convinces the verifier of the statement's truth, marking a critical advancement in cryptographic efficiency. Groth's work, in particular, has been instrumental in the development of efficient and concise non-interactive arguments, commonly referred to as succinct non-interactive arguments (SNARGs), and the closely related succinct non-interactive arguments of knowledge (SNARKs). Notably, many of these cutting-edge SNARGs leverage pairing-based cryptography, as elucidated in Groth's research. As the adoption of zero-knowledge proofs continues to gain momentum, their applications span knowledge verification, and notably, they play an integral role in enhancing the scalability of blockchain technology, underlining their growing significance in the world of cryptography and distributed systems.

## **2.9 Decentralized Society: Finding Web3's Soul.**

**Authors: E. Glen Weyl, Puja Ohlhaver, Vitalik Buterin.**

The paper explores the concept of a decentralized society and its implications for the development of Web3 technologies. The author argues that the current centralized nature of the internet has led to a number of problems, including data breaches, censorship, and the concentration of power in the hands of a few tech giants. The paper then presents a vision for a decentralized society, in which power is distributed among individuals and communities through the use of blockchain and other Web3 technologies. The author discusses the challenges and opportunities of building such a society, including the need for new

governance structures and the importance of user privacy and data ownership. The paper also highlights some of the current projects and initiatives that are working towards this vision, including decentralized social networks and alternative financial systems. Overall, the paper provides a thought-provoking exploration of the potential of Web3 to create a more decentralized, democratic, and equitable society.

## **2.10 EIP721: ERC-721 Non-Fungible Token Standard.**

**Authors: W. Entriken, D. Shirley, J. Evans, and N. Sachs.**

The paper introduces the ERC-721 non-fungible token standard, which is used in Ethereum blockchain for representing unique and indivisible digital assets. The paper explains the motivation behind the standard, highlighting the limitations of fungible tokens (such as ERC-20) in representing unique assets such as artwork, collectibles, and game items. The paper then outlines the technical specifications of the ERC-721 standard, including the structure of the token contract, the functions for creating, transferring, and querying tokens, and the events for tracking token ownership and state changes. The paper also discusses some of the design considerations and trade-offs involved in creating a non-fungible token standard, such as the need for gas optimization, the trade-off between simplicity and flexibility, and the challenges of implementing a standardized interface across different use cases. Overall, the paper provides a comprehensive introduction to the ERC-721 standard, which has become an important building block for a wide range of decentralized applications and digital asset ecosystems on the Ethereum blockchain.

## **2.11 Blockchain in education management: present and future applications**

**Authors: Preeti Bhaskar, Chandan Kumar Tiwari, Amit Joshi.**

This paper discusses the potential applications of blockchain technology in the field of education management. The authors present a review of the current challenges in education management and the ways in which blockchain can be used to address these challenges. The paper also highlights several existing blockchain-based education projects and explores the potential future developments in this area. The authors suggest that blockchain can be used for secure storage of academic records, certification, and verification of qualifications, enabling peer-to-peer learning and collaboration, and improving the transparency and accountability of educational institutions.

## 2.12 COMPARATIVE CASE STUDY

Table 1. Comparative Case Study

<i>Literature Review</i>			
<b>Author</b>	<b>Title</b>	<b>Implementation</b>	<b>Findings</b>
Noni Naor et al.[1]	Pricing via processing, or, combatting junk mail, advances in cryptology.	The invention of Proof of Work(Pow) to reduce spamming email using cryptography.	PoW
Satoshi Nakamoto[2]	Bitcoin: A Peer-to-Peer Electronic Cash System	World First Successful Open Blockchain using cryptography and PoW Consensus mechanism.	Bitcoin(BTC) and Blockchain
Charlie Lee[3]	Litecoin White Paper	A Bitcoin Forked p2p blockchain with improved transaction speed.	Litecoin
Sunny King and Scott Nadal[4]	Ppcoin: Peer-to-peer crypto-currency with proof-of-stake.	First p2p blockchain with Proof of Stake(PoS) consensus mechanism	PoS
Nick Szabo[5]	Secure Property Titles with Owner Authority	An idea to secure assets by implementing smart contracts.	Smart Contract
Vitalik Buterin[6]	Ethereum White Paper	World's First Programmable Open Blockchain which acts similar to a decentralized database.	Ethereum blockchain, EVM, and ETH
Billy Markus et al.[7]	Dogecoin Whitepaper	A Meme Based Blockchain based on Litecoin fork.	DogeCoin
Sandeep Nailwal et al.[8]	The MATIC Network	A successful layer2 blockchain that focuses on ethereum scalability and high gas problems.	MATIC, EVM compatible
Vitalik Buterin et al.[9]	EIP-20: Token Standard	Ethereum's ERC20 Token Standard for making fungible token	ERC20
Ryoshi[10]	SHIBA INU WoofPaper	A Meme Based ERC20 Token and the world's first largest crypto as well as a covid fund to India.	SHIB Token, Leash, Bone, and Shibarium
William Entriken et al.[11]	EIP-721: Non-Fungible Token Standard	Ethereum's ERC721 Token Standard for making non-fungible tokens.	ERC721
AGreg Solano et al.[12]	Bored Ape Yacht Club	World's Most Successful NFT Digital Assets.	10000 Bored Ape Digital Arts, and ApeCoin
Vitalik Buterin et al.[13]	Decentralized Society: Finding Web3's Soul	A proposal of SoulBound Tokens that can be attached to the user's wallet.	SBT
Jamesbachini[14]	Solidity-SBT-SoulBound-Token	An experiment in creating a soul bound token (SBT)	Experiment of NTT



Web3.storage Team[15]	Web3 Storage - Simple file storage with IPFS and Filecoin	A framework to implement and deploy data in IPFS.	web3.storage
Aaron Davis et al.[16]	Metamask	The crypto wallet for Defi, Web3 Dapps and NFTs.	ECC Wallet

In Blockchain technology, there is a trustworthy way to handle these digital assets, and these are programmed with smart contracts and tokens. This paper is highly based on Soul Bound Tokens which can be used in decentralizing digital certificates.

A token is a form of value and it may also represent information. The popular token protocols are ERC20 and ERC721. The ERC20 tokens are often considered utility values, similar to the cryptocurrencies -bitcoin(BTC) and ethereum(ETH). The major difference between ERC20 tokens and cryptocurrencies is ETH, and BTC is native cryptocurrencies(money) of the native blockchains. But the ERC20 is a money-like token that can function inside the ethereum main net. Shiba Inu is an ERC20 token and DogeCoin is a litecoin forked native cryptocurrency.

Similarly, the ERC721 - Non-Fungible Tokens are tokens, that can represent digital and physical assets. The BTC, ETH, SHIB, and DOGE are fungible in nature. The 1 BTC represents 1 BTC, i.e all the BTCs have the same characteristics and properties. On the other hand, Non-Fungible like Bored Ape Yacht Club's Apes are unique and every ape has its own characteristics and properties. In the real world, every property is unique and it always has its own characteristics. We can say that Non-Fungible APE1 is not equal to APE2, similarly, One House property is not equal to Second House property. The major difference between ERC20 and ERC721 is divisible and non-divisible. Both standards are tradable in nature. In our problem, one should not trade or sell his/her certificate token with others, and it may cause serious problems. And also, One should not divide or share his/her certificate ownership with others. These drawbacks are to be considered, making a novel non-transferable token with few outstanding features of ERC20 and ERC721.

These foundational elements collectively shape the backdrop against which PoKoC emerges as a groundbreaking solution, not only within the blockchain domain but also as a catalyst for addressing real-world challenges in a knowledge-centric society.

## **CHAPTER 3**

## **CHAPTER 3**

### **EXISTING SYSTEM**

#### **3.1 Password based Credential Authentication Systems**

Traditional password based credential authentication systems is a widely used authentication system that relies on users' passwords to grant access to resources.

However, the password-based authentication system has several weaknesses that make it vulnerable to attacks. For example:

Passwords can be easily guessed or cracked: Many users choose weak passwords that are easy to guess or crack using brute-force attacks. This makes it easy for attackers to gain unauthorized access to resources.

Password reuse: Many users reuse the same password across multiple accounts and services, which means that if an attacker gains access to one account, they can potentially access others as well.

Social engineering attacks: Attackers can use social engineering tactics, such as phishing or pretexting, to trick users into revealing their passwords or other sensitive information.

Passwords can be intercepted: Passwords can be intercepted by attackers using various methods, such as keyloggers or packet sniffers, when they are entered into a website or application.

Lack of two-factor authentication: Password-based authentication systems often lack two-factor authentication, which can make them vulnerable to attacks that exploit stolen or guessed passwords.

These weaknesses make password-based authentication systems vulnerable to a wide range of attacks, and many organizations are now exploring alternative authentication systems, such as biometric authentication and multi-factor authentication, to improve security.

#### **3.2 OpenID Connect (OIDC) Protocol[25]**

One example of an existing credential authentication system that is not using blockchain and zero-knowledge proof is the OpenID Connect (OIDC) protocol. OIDC is an

open standard for authentication and authorization that is widely used by companies and organizations to manage user identities and credentials.

OIDC works by enabling users to log in to a website or application using their existing credentials from a trusted identity provider, such as Google or Facebook. When a user logs in, the identity provider verifies their identity and issues an access token, which the user can then use to access other services and resources.

OIDC is not based on blockchain technology or zero-knowledge proof, but rather on a centralized model of authentication and authorization. This means that users have to trust the identity provider to manage their credentials and ensure that they are secure and accurate.

Despite its limitations, OIDC has become a popular and widely adopted standard for managing user identities and credentials. Its simplicity and ease of use have made it an attractive option for companies and organizations looking to streamline their authentication and authorization processes.

### **3.3 Security Assertion Markup Language (SAML) Protocol[26]**

Another example of an existing credential authentication system that is not using blockchain and zero-knowledge proof is the Security Assertion Markup Language (SAML) protocol. SAML is an XML-based standard for exchanging authentication and authorization data between parties, such as web applications and identity providers.

SAML works by allowing users to log in to a web application using their existing credentials from a trusted identity provider. When a user logs in, the identity provider issues a SAML assertion, which contains information about the user's identity and the permissions they have been granted. This assertion is then sent to the web application, which can use it to authenticate the user and grant access to the requested resources.

SAML is not based on blockchain technology or zero-knowledge proof, but rather on a centralized model of authentication and authorization. This means that users have to trust the identity provider to manage their credentials and ensure that they are secure and accurate.

Despite its limitations, SAML has become a widely adopted standard for managing user identities and credentials in enterprise environments. Its flexibility and interoperability

have made it an attractive option for companies and organizations looking to integrate their various systems and applications.

### **3.4 Drawbacks of Existing Systems**

Existing credential authentication systems that are not using blockchain and zero-knowledge proof will have the following drawbacks:

**Centralized:** Many existing credential authentication systems are centralized, which means that they rely on a single trusted entity to verify and manage credentials. This creates a single point of failure and makes the system vulnerable to hacking, data breaches, and other security threats.

**Privacy concerns:** Traditional credential authentication systems often require users to share sensitive personal information with third-party organizations, which raises privacy concerns. This information can be misused or stolen, and users have little control over how it is used.

**Lack of transparency:** Centralized credential authentication systems often lack transparency and accountability. Users have to trust the central authority to manage their credentials and ensure that they are secure and accurate.

**Limited interoperability:** Different organizations may use different credential authentication systems, which can create interoperability issues. This can make it difficult for users to share their credentials across different platforms and services.

**Inefficiency:** Traditional credential authentication systems can be slow and inefficient, particularly when it comes to verifying the authenticity of large numbers of credentials. This can create bottlenecks and delays in the credentialing process, which can be frustrating for users.

**High costs:** Traditional credential authentication systems can be expensive to maintain, particularly for large organizations that need to manage a large number of credentials. This can create a financial burden for both organizations and users.

## **CHAPTER 4**

## **CHAPTER 4**

### **PROPOSED SYSTEM**

In this value-fueled economy, the proofs are considered facts. These proofs can also be modified/reproduced by any intruder. To possibly decrease intruder fraud, there should be an open and secure system, and also it should be trustworthy for the participant of this system. These systems often correspond with blockchain, decentralized technology, or emerging web3. What is the value? Value means the usefulness of something. In this work, we consider digital certificates as the value of knowledge proof. Let's get into Alice and Bob's high school. Alice always scores better grades in her class. Poor Bob has a low profile. One day Bob plans to cheat on his certificate. He is left with two options. The first is to find a person who can provide him with a fake original-looking certificate. Second, He somehow managed to copy the profile details of his classmate Alice. Let us convert this into a digital problem, where Alice and Bob have digital certificates. But now the problem even gets worse, hence digital entities can be easy to copy/reproduce. What if there is a solid way to verify that the digital certificates are only to be produced by the high school for Alice and Bob? Either Alice, Bob, or anyone cannot change this transaction. This evolves into the idea of non-transferable proofs. It is like the identity proof of the person and is always attached to him/her.

#### **4.1 Algorithm**

This paper introduces the PoKoC algorithm with a Multi-Signature Scheme. It focuses on two types of knowledge: private and public. Private knowledge is represented using Zero Knowledge Proofs, while public knowledge employs open verifiable proofs. Both types are implemented within an immutable, non-transferable ownership framework. On-chain data includes knowledge hashes, JSON data, and supporting context securely stored in smart contract storage. The Multi-Signature Scheme enhances existing ECC schemes, adding a layer of security to authenticate knowledge among designated parties. The PoKoC algorithm is a modified version of the ERC721 standard and integrates ZKSNARKS, all within the scope of this research.

Think of PoKoC with Multi-Signature Scheme as a digital vault for your knowledge credentials:

**Private Credentials:** Imagine this as your secret knowledge. Only you can prove this knowledge to the verifier, ensuring your privacy. Example: You are qualified in a specific knowledge field, you can prove your knowledge without exposing any other knowledge data.

**Public Credentials:** These are also verifiable knowledge. They are immutable and safely stored on the decentralized ledger, allowing anyone to view and verify them. Example: Your digital certificates, eBadges, etc. In both cases, "Multi-Signature" refers to the authentication process involving multiple authorities.

However, as we progress through this paper, we will unveil an innovative solution – the PoKoC algorithm with a Multi-Signature Scheme. By focusing on private and public knowledge, this algorithm brings the promise of privacy, immutability, and transparency to a modular system, all within a secure, decentralized ledger. Join us in the following sections as we delve deeper into the intricacies of PoKoC, exploring its components, implementation, and its potential to reshape the landscape of knowledge verification in our dynamic digital world.

#### **4.1.1 Motivation**

Our research was prompted by the need for secure, versatile, and auditable non-transferable tokens (NTTs) on the EVM blockchain. We aimed to address critical issues surrounding the security, transparency, and utility of NFTs as NTTs, especially in contexts requiring multi-signature (multi-sig) capabilities and privacy-preserving mechanisms. To achieve this, we developed the PoKoC model, which incorporates pluggable ZKP-Groth16 (Zoraktes), enhancing the ERC721 standard

#### **4.2 The Overall Design of the Proposed System.**

The following diagram abstractly explains the overall design of the system



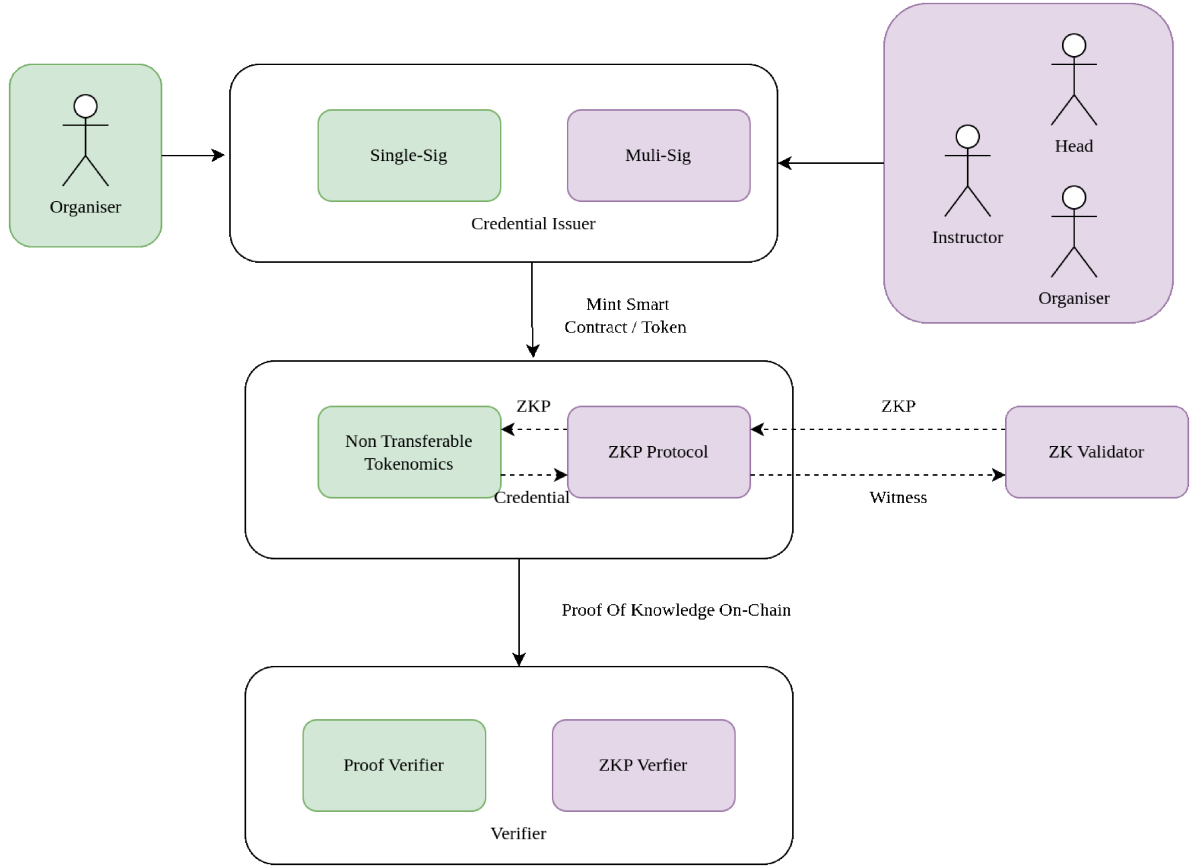


Figure 1 : The Overall Architecture of the Proposed System

### 4.3 Methodology

In the development of the Proof of Knowledge on Chain (PoKoC) system, we employ a comprehensive approach encompassing various components, with particular emphasis on the Non-Transferable Token (NTT) smart contract, the Multi-Signature Scheme, and Groth16 zk-SNARKs. This methodology ensures a secure and tamper-proof framework for knowledge verification within the blockchain ecosystem.

### 4.4 Non-Transferable Token

The NTT smart contract extends the ERC721 standard, introducing a fundamental element of the PoKoC system. This smart contract, implemented in Solidity, is designed to create tokens that are explicitly non-transferable. In contrast to standard ERC721 tokens, which can be freely exchanged between users, NTT tokens are minted by the sender and exclusively owned by a predefined recipient address. This recipient address is immutable, established during contract deployment, ensuring the tokens remain under the control of the intended recipient without the possibility of transfer.

## 4.5 Multi-Signature Scheme

To fortify security and control over NTT tokens, we implement a Multi-Signature Scheme as a critical component of the PoKoC system. This scheme mandates the collective approval of specific actions, such as token minting, by multiple authorized parties. Key aspects of the Multi-Signature Scheme include:

**Authorization:** A registry of authorized addresses is maintained within the contract, enabling the contract operator (typically the deployer) to manage authorized participants. Only listed addresses can engage in the Multi-Signature Scheme.

**Threshold Validation:** Before executing crucial actions, the contract validates whether the action meets the predefined threshold for authorization. This threshold stipulates the minimum number of required authorizations, preventing unilateral control over tokens by any single entity.

**Signature Verification:** The contract verifies whether a specific address has already authorized a particular action, thereby preventing duplicate authorizations and preserving the integrity of the Multi-Signature Scheme

## 4.6 Proof Of Knowledge

The core of the PoKoC system's privacy and security lies in the integration of Groth16 zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge), implemented with Zokrates. This addition empowers the creation of non-polynomial time verifiable proofs, attesting to specific knowledge without disclosing the actual information. Groth16 zk-SNARKs enable one party (the prover) to prove to another party (the verifier) that they possess specific knowledge without revealing the knowledge itself. Key advantages of Groth16 zk-SNARKs within the PoKoC system include:

- **Privacy:** Users can prove ownership or authorization without divulging sensitive information, such as private keys or personal data.
- **Efficiency:** Groth16 zk-SNARKs offer a highly efficient method for generating and verifying proofs, reducing computational costs.
- **Scalability:** The utilization of zk-SNARKs facilitates the scalability of the PoKoC system while maintaining robust security guarantees.

## **4.7 Public Credentials**

PoKoC incorporates regular NTT credentials, which are publicly viewable, allowing users to demonstrate ownership or authorization without revealing sensitive information. These regular NTT credentials serve as a fundamental component of the PoKoC system, ensuring transparency, accessibility, and immutability.

## **4.8 Private Credentials & Hashed Proofs**

Within the NTT smart contract, private credentials, such as ownership or authorization, undergo validation using Groth16 zk-SNARKs. These non-polynomial time verifiable proofs are generated by the prover, demonstrating their knowledge of specific information without disclosing the information to the verifier. These generated Groth16 zk-SNARK proofs can be stored as hashes within the context of the URI associated with NTT tokens storage. This innovative approach ensures efficient and secure private credential verification while preserving user privacy.

The PoKoC system integrates NTT, the Multi-Signature Scheme, and Groth16 zk-SNARKs to establish a robust framework for knowledge verification and educational credentials. This enhances security, privacy, and scalability, making PoKoC an effective solution for various knowledge-centric domains.

## 4.9 Architectural View

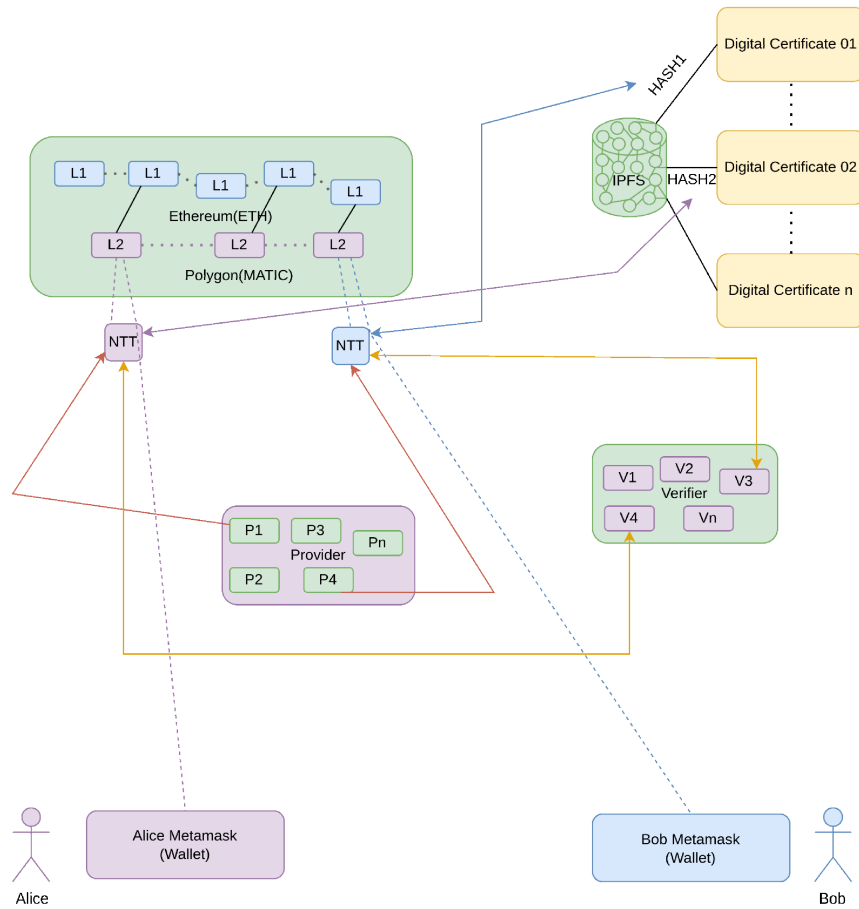


Figure 2 : Architecture of Non-Transferable Tokenomics

From Figure 2, We can understand the overall architecture of the system. Either Alice, Bob, or anyone can not proxy the identity of themselves and their data. The L1s are layer1 ethereum and L2s are layer2 polygon. We can understand that IPFS is decently decentralized by the nodes. The data and hashes are stored in IPFS. We also understand the connectivity between metamask, providers, verifiers, NTT, IPFS, and the blockchain.

## 4.10 Advantages Of PoKoC

1. Non-Transferable Tokonomics: PoKoC introduces NTT tokens, which are explicitly non-transferable, ensuring that ownership remains with the intended recipient. This unique tokenomic model enhances security and control over educational credentials.
2. Highly Decentralized: PoKoC leverages the decentralized nature of blockchain technology, reducing the reliance on centralized authorities for knowledge verification. This decentralization enhances trust and transparency.

3. **Math-Based Solution:** The integration of Groth16 zk-SNARKs provides a mathematically rigorous method for knowledge verification. It allows one party to prove knowledge without revealing sensitive information, enhancing privacy and security.
4. **Sue the Proxy:** PoKoC includes a Multi-Signature Scheme, requiring collective authorization for critical actions. This feature prevents unilateral control and misuse of educational credentials.
5. **Low Gas Fee:** The PoKoC system is designed to maintain low gas fees, ensuring cost-effectiveness for users while still providing robust security measures.
6. **Sustainable Computing:** By reducing computational costs through efficient zk-SNARKs, PoKoC contributes to sustainable computing practices within the blockchain ecosystem.
7. **Easy Integration:** PoKoC is adaptable and can be seamlessly integrated into various use cases, making it a versatile solution for knowledge-centric domains.
8. **Overall,** PoKoC's combination of NTT tokens, Multi-Signature Scheme, and Groth16 zk-SNARKs establishes a secure, privacy-focused, and scalable framework for knowledge verification and educational credentials.

#### **4.11 Summary**

The PoKoC system combines NTT, the Multi-Signature Scheme, and Groth16 zk-SNARKs to create a robust framework for secure, private, and scalable knowledge verification and educational credentials. This innovative approach aligns with the growing demand for censorship-resistant identities in the age of public blockchains and decentralization. It empowers individuals to safeguard their valuable data and personal reputation in the digital space, laying the foundation for Non-Transferable Tokenomics. By decentralizing digital certificates through blockchain technology, PoKoC enhances security and trust in knowledge-centric domains.

## **CHAPTER 5**

# CHAPTER 5

## SYSTEM SPECIFICATION

### Hardware Specification

PROCESSOR	:	INTEL CORE I7
RAM	:	8.00 GB
STORAGE CAPACITY	:	500 GiB

### Software Specification

OPERATING SYSTEM	:	LINUX
LANGUAGE	:	Solidity, JavaScript
LIBRARIES	:	Zokrates: Groth16
IDE	:	Remix, VS Code
FrontEnd	:	Ether.js, React.js
Backend	:	Ethereum Blockchain and Polygon Blockchain

## **CHAPTER 6**



## **CHAPTER 6**

### **SOFTWARE TECHNOLOGIES USED**

The following are some of the important software technologies used in the design of the proposed system.

#### **6.1 Solidity**

Solidity is a programming language used for writing smart contracts on the Ethereum blockchain. It was developed in 2014 by a group of developers led by Gavin Wood. Solidity is a statically typed language, meaning that the data type of a variable is declared at the time of its creation and cannot be changed later. Smart contracts are self-executing programs that run on a blockchain, and Solidity is the language used to write these contracts on the Ethereum network. These contracts contain a set of rules and conditions that are automatically executed when certain criteria are met.

Solidity is similar to other programming languages like JavaScript and C++, making it relatively easy for developers to learn. However, there are some unique aspects of Solidity that require additional understanding, such as the use of gas fees to execute code on the Ethereum network and the fact that all code is publicly visible on the blockchain. Overall, Solidity is an essential tool for developers looking to build decentralized applications (dApps) on the Ethereum network. It allows developers to create secure and transparent smart contracts that can be executed automatically, providing a new level of trust and transparency to online transactions.

#### **6.2 ZK-Snark**

Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (ZK-SNARK) is a type of cryptographic proof system that allows one party (the prover) to prove to another party (the verifier) that a statement is true without revealing any additional information beyond the truth of the statement itself. ZK-SNARK is used to provide privacy and confidentiality in blockchain transactions. It enables users to verify transactions without revealing any sensitive information about the parties involved or the details of the transaction. This technology has been used in cryptocurrencies like Zcash to provide an additional layer of privacy to its users. ZK-SNARKs work by creating a proof that a statement is true, without revealing any information about the statement itself. This proof is then verified by the other

party, who can be sure that the statement is true without having any additional information. The proof is created using complex mathematical algorithms, and the verification process is very fast, allowing transactions to be processed quickly and efficiently. The use of ZK-SNARKs in blockchain technology has important implications for privacy and security. It allows for anonymous transactions without the need for third-party intermediaries or trusted authorities. This technology could be used in a variety of applications beyond cryptocurrencies, including voting systems, supply chain management, and identity verification.

However, the implementation of ZK-SNARKs can be complex and requires a deep understanding of cryptography and mathematical algorithms. As with any new technology, there are also potential risks and vulnerabilities that need to be carefully considered and addressed.

### **6.3 Zokrates**

Zokrates is an open-source toolbox essential for blockchain developers, simplifying the implementation of zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge). With its domain-specific language (DSL), constraint system, and compilation capabilities, Zokrates empowers developers to create and verify zk-SNARKs efficiently without delving into intricate cryptographic details. It plays a pivotal role in privacy-preserving and scalable blockchain applications by enabling the generation of proofs that validate specific computations or statements without revealing sensitive data. Zokrates is commonly integrated with Ethereum smart contracts, contributing to enhanced privacy and efficiency in blockchain transactions, making it a valuable asset in the blockchain development toolkit.

### **6.4 Ethers.js**

The library includes a range of functions for interacting with the Ethereum network, such as querying account balances, sending transactions, and interacting with smart contracts. It also includes a number of additional features, such as support for the Web3 API, which allows developers to connect to the Ethereum network and interact with it using JavaScript. Ethers.js is open-source and can be used by developers working on both front-end and back-end applications. It is also designed to work seamlessly with other Ethereum development tools, such as Remix, Truffle, and Ganache. One of the key benefits of using Ethers.js is that

it allows developers to write code in a familiar language, JavaScript, which is widely used in web development. This makes it easier for developers to get started with building dApps on the Ethereum network, as they do not need to learn a new programming language from scratch.

Overall, Ether.js is a powerful tool for developers looking to build decentralized applications on the Ethereum network. It provides a range of tools and utilities that make it easier to interact with the blockchain, and it is widely used and supported by the Ethereum community.

## **6.5 SolidityScan**

SolidityScan is a fundamental tool in the realm of smart contract security assessment, offering a comprehensive approach to categorizing vulnerabilities based on their severity levels, thus providing a clear understanding of potential security risks. The severity levels, ranging from Critical to Informational, cover a wide spectrum of vulnerabilities, from those with the potential to fully compromise a contract's security (Critical) to issues that may indicate coding practice problems (Informational). Additionally, SolidityScan places a special emphasis on optimizing gas usage through its Gas category, contributing to cost-effective contract execution and reduced transaction fees.

What sets SolidityScan apart is its calculation of an overall security score, which takes into account the severity level of each identified issue. By employing a severity-weighted formula, this score provides a holistic evaluation of smart contract security, considering not only the presence of vulnerabilities but also their potential impact on the contract's security. This multi-dimensional approach ensures that security assessments extend beyond mere identification of weaknesses to a deeper understanding of their significance in the broader context of the contract's security posture.

## **CHAPTER 7**

## CHAPTER 7

### DESIGN OF POKOC

In the Design of PoKoC, we delve into the intricate processes that govern the interaction between the Authoritative Sender and the Knowledgeable Recipient, all orchestrated within the deployed Contract. These processes are rooted in cryptographic principles and verification mechanisms that ensure the secure and tamper-proof nature of the system. To grasp the essence of PoKoC's architecture and its verification mechanisms, we present an abstract view of the system's flow and delve into the intricacies of verifying both public and private proof data. This section sheds light on the robust foundation upon which PoKoC stands, emphasizing security and transparency in knowledge verification.

The Authoritative Sender  $A(S)$  deploying the EVM entering Contract  $e(C)$ , it being deployed to the Knowledgeable Recipient  $K(R)$  as deployed Contract  $d(C)$ , where  $I$  is the identity element of any address over the ECC Finite Field  $\mathbb{F}$  with the property  $S.I = S$ :

$$A(S) \rightarrow e(C) \rightarrow K(R) \rightarrow d(C) \quad (1)$$

In general, the verification function  $V$  for a given message  $m$  and a digital signature  $s$  returns the sender's address  $S$  if the verification is successful, where  $S$  and  $s$  belong to the Finite Field  $\mathbb{F}$ :

$$V(S, m, s) \rightarrow S : S, s \in \mathbb{F} \quad (2)$$

Interaction between  $A(S)$  and Deployed Contract  $d(C)$  to update the registry  $R$  of Authoritarians  $A_1, A_2, A_3, \dots, A_n$ :

$$A(S) \xrightarrow{d(C)} R(A_1, A_2, A_3, \dots, A_n) \quad (3)$$

$R(A)$  Registered Authoritarian uploads the NTT data  $D(T)$  in the deployed Contract  $d(C)$ :

$$R(A) \xrightarrow{D(T)} d(C) \quad (4)$$

Signatures of the Registered Authoriatirans  $R(A_1, A_2, A_3, \dots, A_n)$  to collectively indicate the threshold in the contract  $t(C)$  as the sum of Signatures  $\Sigma_S$ :

$$R(A_1, A_2, A_3, \dots, A_n) \xrightarrow{\Sigma_S} t(C) \quad (5)$$

If the threshold  $t(C)$  in the contract  $d(C)$ , belonging to  $K(R)$ , is met, it can mint the Non-Transferable Token with Data  $NTT(D(T))$ :

$$t(C) \xrightarrow{d(C) \in K(R)} NTT(D(T)) \quad (6)$$

## 7.1 Graphical Design of PoKoC

The following diagram illustrates how  $A(S)$  and  $X(I)$  interact with  $K(R)$  and  $d(C)$  to ensure knowledge, immutability, tamper-proofing, and the Directed Acyclic Property in the context of NTT.

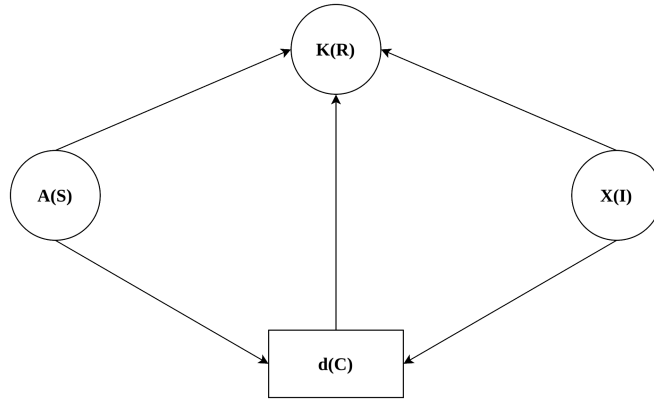


Figure 3 : Directed Acyclic Property of PoKoC

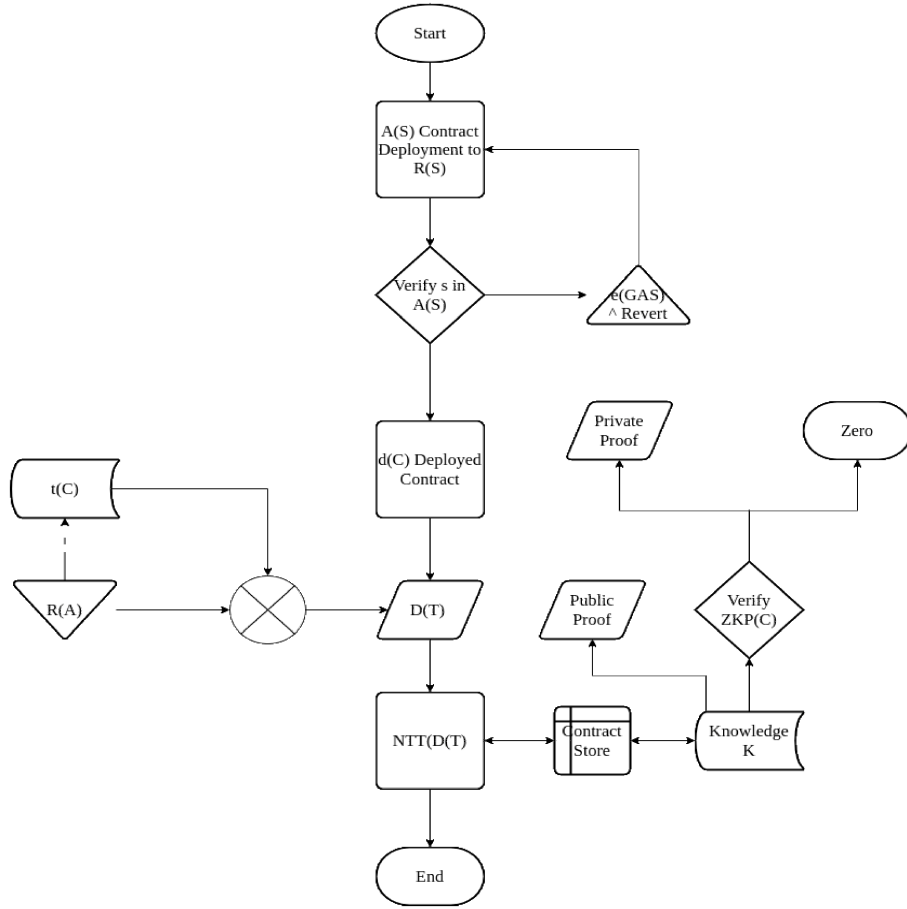


Figure 4 : Design Flow of PoKoC

The PoKoC flow diagram involves the deployment of an authoritative sender  $A(S)$  entering a contract  $e(C)$ , which is deployed to a knowledgeable recipient  $K(R)$  as a deployed contract  $d(C)$ . If certain conditions are met,  $K(R)$  can mint a token  $NTT(D(T))$ .

## 7.2 Verification of PoKoC

**Verification of Public Proof  $P(K)$**  - Anyone can verify  $V$  the public proof data  $P(K)$  using information from  $K(R)$  and  $d(C)$  along with  $NTT(D(T))$  data, which includes JSON data and other URI data.

$$V(P(K)) = f(K(R), d(C), NTT(D(T))) \quad (7)$$

**Verification of Private Proof  $P(Z)$**  - Anyone can verify the private zero-knowledge proof  $P(Z)$  using the hash  $H(K)$ , which is derived from the zero-knowledge verifiable contract.

$$V(P(Z)) = g(H(K)) \quad (8)$$

In summary, the Design of PoKoC represents a sophisticated system built upon cryptographic principles and verification mechanisms, offering a robust foundation for secure and tamper-proof knowledge verification. The interaction between the Authoritative Sender, Knowledgeable Recipient, and the deployed Contract ensures the integrity and immutability of the system. With the ability to verify both public and private proof data, PoKoC provides a comprehensive solution for knowledge credential verification.

The described contract life-cycle, from updating the registry of Authoritarians to the issuance of NTT tokens, demonstrates the system's reliability and security. The collective signatures of Authoritarians and the threshold mechanism add an additional layer of trust and security to the process.

The verification of PoKoC, both for public proof and private zero-knowledge proof, ensures that the system can be transparently audited and validated by any interested party. Overall, PoKoC offers a pioneering approach to secure knowledge credential verification, with a strong emphasis on security, transparency, and efficiency, making it a valuable addition to the world of decentralized ledger systems.



## **CHAPTER 8**

## CHAPTER 8

### RESULTS AND DISCUSSION

In this section, we make a comprehensive discussion of our work, focusing on the innovation of our PoKoC within the ERC721 standard. We elucidate the motivations behind our research, the rigorous testing conducted, and the unique features that distinguish our approach from traditional ERC721 implementations.

#### 8.1 Metrics Used for Evaluation

It's essential to highlight the security assessment aspect of our work. We employed SolidityScan, a robust system that conducts both static and dynamic analysis of smart contracts(SolidityScan . 2023). We used SolidityScan as a metric to measure the security components of the algorithms, under investigation. This analysis yielded a security score, which serves as a quantitative measure of a smart contract's vulnerability level. This score takes into account factors such as vulnerabilities detected by static analysis, severity of vulnerabilities, contract complexity, and contract popularity.

#### 8.2 Security Levels

SolidityScan categorizes vulnerabilities into distinct severity levels, each indicative of the potential impact on the contract's security:

- **Critical:** Signifying vulnerabilities with the capacity to fully compromise the contract's security.
- **High:** Denoting vulnerabilities that could lead to significant financial losses or substantial damages.
- **Medium:** Covering vulnerabilities that may result in minor financial losses or moderate damages.
- **Low:** Representing vulnerabilities with a low probability of exploitation but warranting attention.
- **Informational:** Encompassing vulnerabilities that do not pose immediate security risks but may indicate coding practice issues.

- Gas: Focusing on optimizing gas usage for cost-effective contract execution and lower transaction fees.

### 8.3 Overall Security Score Calculation

The overall security score is a comprehensive measure that considers the severity level of each identified issue. It factors in issues categorized under various severity levels, including Critical, High, Medium, Low, Informational, and Gas Usage. The calculation employs a severity-weighted formula, as represented below:

$$S = W1 \cdot I_h + W2 \cdot I_m + W3 \cdot I_l + W4 \cdot I_i + W5 \cdot I_g \quad (9)$$

$S$  : Overall Security Score

$W1, W2, W3, W4, W5$  : Weight factors for issues

$I_h, I_m, I_l, I_i, I_g$  : Counts of issues (High, Med, Low, Info, Gas)

This formula assigns appropriate weights to each severity level and multiplies them by the number of issues detected at that specific severity level. The resulting values are then aggregated to produce the overall security score. A higher score denotes a higher level of security, providing a thorough assessment of the smart contract's security posture.

### 8.4 5.2 Tables and Graphs Analysis

In this subsection, we discuss on a comprehensive technical examination of the tables, graphs, and key observations derived from the implementation of our groundbreaking PoKoC model. These analytical components hold significant importance in elucidating the elevated levels of security and reliability that our model introduces to the realm of decentralized ledgers and tamper-proof knowledge management systems.

In Table 1 and Fig3, we present a comparative analysis of security audit scores across various ERC721 models. These scores serve as direct indicators of the security and reliability profiles of each model:

**Table 2. Security Audit Scores**

Model	Score
Minimum ERC721	51.79
Standard ERC721	53.85
Extended ERC721	60.47
PoKoC ERC721	71.72

**Minimum ERC721 (51.79):** This benchmark corresponds to the most fundamental implementation of the ERC721 standard, establishing the core foundation for non-fungible token (NFT) contracts.

**Standard ERC721 (53.85):** Derived from the reputable OpenZeppelin library, this model adheres rigorously to established industry standards and best practices, reflecting a solid security posture.

**Extended ERC721 (60.47):** Tailored for a specific blockchain application (QuickNode), this extended version of the ERC721 standard incorporates additional features meticulously crafted to fulfill the unique requirements of this application.

**PoKoC ERC721 (71.72):** PoKoC ERC721 stands out with an impressive security score of 71.72. Its distinctive feature is the incorporation of multi-signature (multi-sig) support, purpose-built for non-transferable tokens (NTT). This innovation enables multiple authors to collaboratively issue tokens to a single recipient, significantly enhancing security and trust. The notably high security score attests to its robustness and reliability when compared to other models.

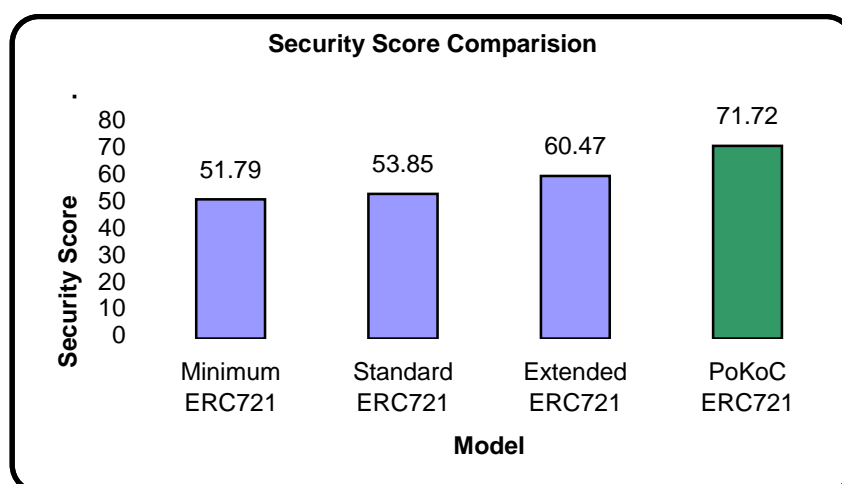


Figure 5 : Security Score Comparison

Figure 5 graph represents comparison of the security scores of various implementations, highlighting the superior security of PoKoC ERC721 with a score of 71.72 compared to other models.

Table 2 provides an insightful comparison of security scores, distinguishing between two significant configurations within our PoKoC ecosystem:

**Single Contract (71.72):** This configuration represents a specific contract, essentially serving as an abstracted version of the PoKoC ERC721. It has undergone strategic modifications to seamlessly incorporate Non-Transferable Tokens (NTT), employing a

sophisticated multi-signature (multi-sig) framework. This adaptation yields an impressive security score of 71.72, underscoring its robust security measures.

**Overall System (86.83):** The "Overall System" constitutes the comprehensive embodiment of the ERC721 standard, including the "Single Contract." This holistic representation encapsulates the entire PoKoC system, encompassing all its components, functionalities, and innovations. Remarkably, the "Overall System" attains a significantly elevated security score of 86.83, surpassing the "Single Contract." This substantial difference underscores the superior security and overall quality inherent in the comprehensive PoKoC system.

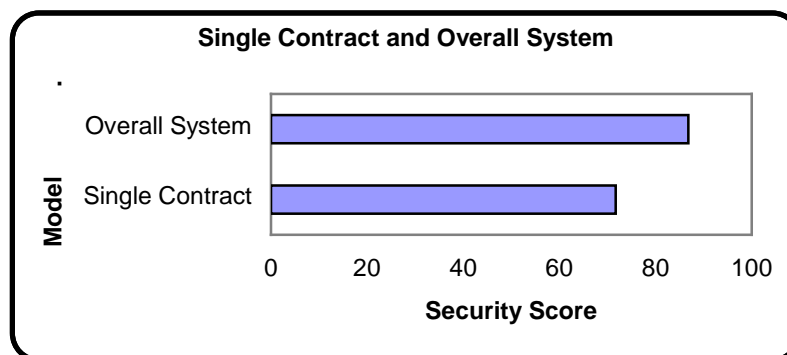


Figure 6 : Single Contract and Overall System

Figure 6 graph represents two configurations: "Single Contract" with a security score of 71.72, incorporating Non-Transferable Tokens (NTT) and multi-signature (multi-sig) features, and the "Overall System" with a higher security score of 86.83, encompassing the entire PoKoC system.

Table 3 and Fig4 provides a comprehensive overview of the security scores for various configurations within our PoKoC system. These scores are categorized based on different severity levels, including Low, Gas, Info, Critical, High, and Med.

**Table 3. System Scores for Different Configurations**

	Low	Gas	Info	Critical	High	Med
Single Contract	6	12	4	0	0	0
Overall System	32	44	9	1	0	0

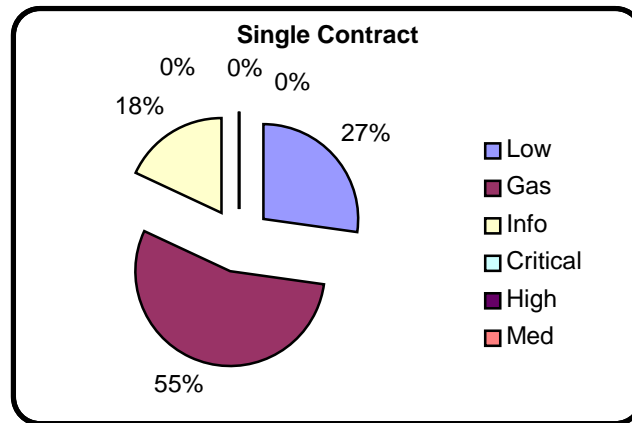


Figure 7 : Security Score of Single Contract

The above pie-chart represents the scores, of the "Single Contract" scoring 6 in the Low, 12 in Gas, 4 in Info, and 0 in Critical, High, and Med Vulnerabilities.

The Fig 5 explains the Security Scores of the "Single Contract" that has 0 Critical, High and Medium Vulnerabilities.

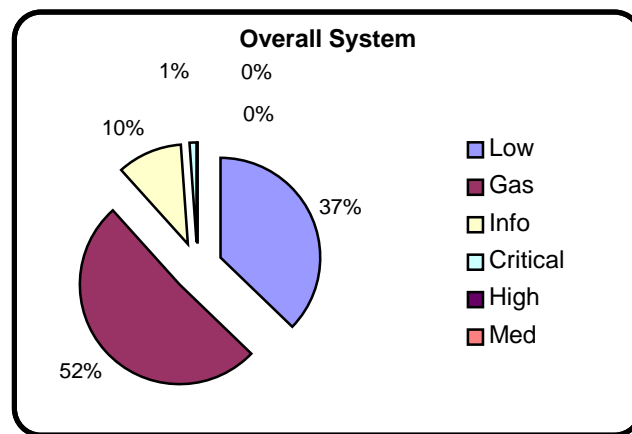


Figure 8 : Security Score of Overall System

The above pie-chart represents the scores, of the "Overall System " scoring 32 in the Low, 44 in Gas, 9 in Info, and 1 in Critical, High, and Med categories.

While the system indeed achieves an impressive overall security score, it's essential to draw attention to a noteworthy discovery – a singular critical issue. This issue arises due to a particular parameter embedded within SolidityScan, which discourages the utilization of similar naming conventions in contracts. It's worth emphasizing that in this context, the "Overall System" pertains to a modified iteration of the ERC721 standard. This adaptation introduces a modularized naming convention, a pivotal adjustment designed to facilitate seamless operation across diverse EVM-enabled ledgers. As a result of this refinement, the critical issue in question was unearthed and is further detailed in Table 3 and Fig 6.

Based on the comprehensive analysis and results presented in the preceding sections, it becomes abundantly clear that our PoKoC algorithm exhibits remarkable robustness when tested in real-world scenarios. Furthermore, the modular framework we have developed demonstrates exceptional versatility, showcasing its capability to seamlessly integrate with an extensive array of systems and applications. These findings underscore the potential of PoKoC to significantly enhance the landscape of secure and decentralized knowledge management, ushering in a new era of trust and reliability across diverse domains.

## **CHAPTER 9**



## CHAPTER 9

### CONCLUSION AND FUTURE WORK

In our extensive research, we have introduced a state-of-the-art Proof of Knowledge On-Chain (PoKoC) algorithm with a registered Multi-Signature Scheme. This innovation addresses the challenge of requiring multiple authorities to endorse a trustworthy credential. While PoKoC has its foundation in blockchain technology, its potential goes far beyond, offering versatile solutions to real-world challenges in our knowledge-driven economy. It effectively resolves significant issues in education, identity verification, supply chain transparency, healthcare, legal contracts, and various other domains, effectively combating problems like educational scams and knowledge frauds. This approach represents a significant stride towards establishing a trustworthy knowledge economy.

At its core, PoKoC serves as a practical and secure framework for verifying and managing knowledge credentials. Its importance extends beyond the blockchain, with the potential to fundamentally change how knowledge is authenticated and trusted in various domains. The integration of Groth16 zk-SNARKs enhances privacy and security within the PoKoC model, expanding its potential applications beyond blockchain and making it relevant in numerous sectors for data privacy and authentication.

Introducing the concept of Non-Transferable Tokens (NTT) represents more than just an innovation; it signifies a fundamental rethinking of digital asset ownership, privacy and control. NTTs usher in a new era of heightened security and trust in token-based systems, impacting areas ranging from digital credentials to supply chain authenticity. Our commitment to open-source development goes beyond mere philosophy; it encourages global collaboration and transparency as we address real-world challenges. PoKoC[27] is available as an open-source project on Git, inviting contributions from a diverse and innovative community.

As we conclude this research, it's essential to highlight the extensive impact of PoKoC. Beyond its origins in blockchain, PoKoC has the potential to revolutionize various aspects of our knowledge-based economy. We envision PoKoC as a crucial element in shaping a more secure, trustworthy, and efficient digital knowledge management systems.

In summary, this research represents not only a technological milestone but also a beacon of hope for addressing critical issues in our increasingly knowledge-dependent society. PoKoC transcends blockchain; it serves as a catalyst for positive change in the world of knowledge verification and beyond. The journey ahead promises exciting innovations and transformative solutions to real-world challenges.

## **APPENDIX**

## APPENDIX – 1

### SAMPLE CODE

#### PoKoC ERC721.sol - Sample of Contract Code in Solidity

```
001 // SPDX-License-Identifier: MIT
002 // OpenZeppelin Contracts (last updated v4.7.0) (token/ERC721/ERC721.sol)
003
004
005 pragma solidity ^0.8.0;
006 // error NotOwner();
007
008 import "./IERC721.sol";
009 import "@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol";
010 import "./IERC721Metadata.sol";
011 import "@openzeppelin/contracts/utils/Address.sol";
012 import "@openzeppelin/contracts/utils/Context.sol";
013 import "@openzeppelin/contracts/utils/Strings.sol";
014 import "@openzeppelin/contracts/utils/Strings.sol";
015 import "@openzeppelin/contracts/utils/introspection/ERC165.sol";
016
017
018
019 /**
020  * @dev Implementation of https://eips.ethereum.org/EIPS/eip-721[ERC721] Non-
    Fungible Token Standard, including
021  * the Metadata extension, but not including the Enumerable extension, which is
    available separately as
022  * {ERC721Enumerable}.
```

```

023 */
024 contract ERC721 is Context, ERC165, IERC721, IERC721Metadata {
025     using Address for address;
026     using Strings for uint256;
027
028     // Token name
029     string private _name;
030
031     // Token symbol
032     string private _symbol;
033
034     // Mapping from token ID to owner address
035     mapping(uint256 => address) private _owners;
036
037     // Mapping owner address to token count
038     mapping(address => uint256) private _balances;
039
040     // Mapping from token ID to approved address
041     mapping(uint256 => address) private _tokenApprovals;
042
043     // Mapping from owner to operator approvals
044     mapping(address => mapping(address => bool)) private _operatorApprovals;
045
046     address public contractOperator;
047
048     mapping(address => bool) public registry;
049
050
051     /**
052      * @dev Initializes the contract by setting a `name` and a `symbol` to the token
053      collection.
054      */
055     constructor(string memory name_, string memory symbol_) {
056         name = name;

```

```

056 symbol = symbol;
057 contractOperator = msg.sender;
058 }
059
060 /**
061  * @dev See {IERC165-supportsInterface}.
062  */
063 function supportsInterface(bytes4 interfaceId) public view virtual override(ERC165,
    IERC165) returns (bool) {
064 return
065     interfaceId == type(IERC721).interfaceId ||
066     interfaceId == type(IERC721Metadata).interfaceId ||
067     super.supportsInterface(interfaceId);
068 }
069
070 /**
071  * @dev See {IERC721-balanceOf}.
072  */
073 function balanceOf(address owner) public view virtual override returns (uint256) {
074 require(owner != address(0), "ERC721: address zero is not a valid owner");
075 return _balances[owner];
076 }
077
078 /**
079  * @dev See {IERC721-ownerOf}.
080  */
081 function ownerOf(uint256 tokenId) public view virtual override returns (address) {
082 address owner = _ownerOf(tokenId);
083 require(owner != address(0), "ERC721: invalid token ID");
084 return owner;
085 }
086
087 /**
088  * @dev See {IERC721Metadata-name}.

```

```

089  */
090  function name() public view virtual override returns (string memory) {
091  return _name;
092  }
093
094  /**
095   * @dev See {IERC721Metadata-symbol}.
096   */
097  function symbol() public view virtual override returns (string memory) {
098  return _symbol;
099  }
100
101  /**
102   * @dev See {IERC721Metadata-tokenURI}.
103   */
104  function tokenURI(uint256 tokenId) public view virtual override returns (string
    memory) {
105  _requireMinted(tokenId);
106
107  string memory baseURI = _baseURI();
108  return bytes(baseURI).length > 0 ? string(abi.encodePacked(baseURI,
    tokenId.toString())) : "";
109  }
110
111  /**
112   * @dev Base URI for computing {tokenURI}. If set, the resulting URI for each
113   * token will be the concatenation of the `baseURI` and the `tokenId`. Empty
114   * by default, can be overridden in child contracts.
115   */
116  function _baseURI() internal view virtual returns (string memory) {
117  return "";
118  }
119
120  /**

```

```

121  * @dev See {IERC721-approve}.
122  */
123  // function approve(address to, uint256 tokenId) public virtual override {
124  //   address owner = ERC721.ownerOf(tokenId);
125  //   require(to != owner, "ERC721: approval to current owner");
126
127  //   require(
128  //     _msgSender() == owner || isApprovedForAll(owner, _msgSender()),
129  //     "ERC721: approve caller is not token owner or approved for all"
130  //   );
131
132  //   _approve(to, tokenId);
133  // }
134
135  /**
136  * @dev See {IERC721-getApproved}.
137  */
138  // function getApproved(uint256 tokenId) public view virtual override returns
    (address) {
139  //   _requireMinted(tokenId);
140
141  //   return _tokenApprovals[tokenId];
142  // }
143
144  /**
145  * @dev See {IERC721-setApprovalForAll}.
146  */
147  // function setApprovalForAll(address operator, bool approved) public virtual override
    {
148  //   _setApprovalForAll(_msgSender(), operator, approved);
149  // }
150
151  /**
152  * @dev See {IERC721-isApprovedForAll}.

```



```

153     */
154     // function isApprovedForAll(address owner, address operator) public view virtual
        override returns (bool) {
155         //     return _operatorApprovals[owner][operator];
156         // }
157
158     /**
159     * @dev See {IERC721-transferFrom}.
160     */
161     // function transferFrom(
162     //     address from,
163     //     address to,
164     //     uint256 tokenId
165     // ) public virtual override {
166     //     //solhint-disable-next-line max-line-length
167     //     require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: caller is not
        token owner or approved");
168
169     //     _transfer(from, to, tokenId);
170     // }
171
172     /**
173     * @dev See {IERC721-safeTransferFrom}.
174     */
175     // function safeTransferFrom(
176     //     address from,
177     //     address to,
178     //     uint256 tokenId
179     // ) public virtual override {
180     //     safeTransferFrom(from, to, tokenId, "");
181     // }
182
183     /**
184     * @dev See {IERC721-safeTransferFrom}.

```

```

185     */
186     // function safeTransferFrom(
187     //     address from,
188     //     address to,
189     //     uint256 tokenId,
190     //     bytes memory data
191     // ) public virtual override {
192     //     require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: caller is not
        token owner or approved");
193     //     _safeTransfer(from, to, tokenId, data);
194     // }
195
196     /**
197     * @dev Safely transfers `tokenId` token from `from` to `to`, checking first that
        contract recipients
198     * are aware of the ERC721 protocol to prevent tokens from being forever locked.
199     *
200     * `data` is additional data, it has no specified format and it is sent in call to `to`.
201     *
202     * This internal function is equivalent to {safeTransferFrom}, and can be used to e.g.
203     * implement alternative mechanisms to perform token transfer, such as signature-
        based.
204     *
205     * Requirements:
206     *
207     * - `from` cannot be the zero address.
208     * - `to` cannot be the zero address.
209     * - `tokenId` token must exist and be owned by `from`.
210     * - If `to` refers to a smart contract, it must implement {IERC721Receiver-
        onERC721Received}, which is called upon a safe transfer.
211     *
212     * Emits a {Transfer} event.
213     */
214     // function _safeTransfer(

```

```

215 // address from,
216 // address to,
217 // uint256 tokenId,
218 // bytes memory data
219 // ) internal virtual {
220 //     _transfer(from, to, tokenId);
221 //     require(_checkOnERC721Received(from, to, tokenId, data), "ERC721: transfer to
        non ERC721Receiver implementer");
222 // }
223
224 /**
225  * @dev Returns the owner of the `tokenId`. Does NOT revert if token doesn't exist
226  */
227 function _ownerOf(uint256 tokenId) internal view virtual returns (address) {
228 return _owners[tokenId];
229 }
230
231 /**
232  * @dev Returns whether `tokenId` exists.
233  *
234  * Tokens can be managed by their owner or approved accounts via {approve} or
        {setApprovalForAll}.
235  *
236  * Tokens start existing when they are minted (`_mint`),
237  * and stop existing when they are burned (`_burn`).
238  */
239 function _exists(uint256 tokenId) internal view virtual returns (bool) {
240 return _ownerOf(tokenId) != address(0);
241 }
242
243 /**
244  * @dev Returns whether `spender` is allowed to manage `tokenId`.
245  *
246  * Requirements:

```

```

247  *
248  * - `tokenId` must exist.
249  */
250  // function _isApprovedOrOwner(address spender, uint256 tokenId) internal view
    virtual returns (bool) {
251  //     address owner = ERC721.ownerOf(tokenId);
252  //     return (spender == owner || isApprovedForAll(owner, spender) ||
        getApproved(tokenId) == spender);
253  // }
254
255  /**
256  * @dev Safely mints `tokenId` and transfers it to `to`.
257  *
258  * Requirements:
259  *
260  * - `tokenId` must not exist.
261  * - If `to` refers to a smart contract, it must implement {IERC721Receiver-
        onERC721Received}, which is called upon a safe transfer.
262  *
263  * Emits a {Transfer} event.
264  */
265  function _safeMint(address to, uint256 tokenId) internal virtual {
266  _safeMint(to, tokenId, "");
267  }
268
269  /**
270  * @dev Same as {xref-ERC721-_safeMint-address-uint256-}[_safeMint`], with an
        additional `data` parameter which is
271  * forwarded in {IERC721Receiver-onERC721Received} to contract recipients.
272  */
273  function _safeMint(
274  address to,
275  uint256 tokenId,
276  bytes memory data

```

```

277     ) internal virtual {
278     _mint(to, tokenId);
279     require(
280         _checkOnERC721Received(contractOperator, to, tokenId, data),
281         "ERC721: transfer to non ERC721Receiver implementer"
282     );
283     }
284
285     function registerUpdate(address _authoritarianAddress, bool access )
286     public returns (bool)
287     {
288         require(msg.sender == contractOperator, "Not OnlyOwner");
289         registry[_authoritarianAddress] = access;
290         return access;
291     }
292 }
293
294 // modifier onlyOwner {
295 //     // require(msg.sender == operator);
296 //     if (msg.sender != contractOperator) revert NotOwner();
297 //     _;
298 // }
299 /**
300  * @dev Mints `tokenId` and transfers it to `to`.
301  *
302  * WARNING: Usage of this method is discouraged, use {_safeMint} whenever
303  * possible
304  *
305  * Requirements:
306  *
307  * - `tokenId` must not exist.
308  * - `to` cannot be the zero address.
309  *
310  * Emits a {Transfer} event.

```

```

310  */
311  function _mint(address to, uint256 tokenId) internal virtual {
312  require(registry[msg.sender] == true, "Not an authoritarian" );
313  require(to != contractOperator, "ERC721: mint to the user address");
314  require(!_exists(tokenId), "ERC721: token already minted");
315
316  _beforeTokenTransfer(contractOperator, to, tokenId, 1);
317
318  // Check that tokenId was not minted by `_beforeTokenTransfer` hook
319  require(!_exists(tokenId), "ERC721: token already minted");
320
321  unchecked {
322    // Will not overflow unless all 2**256 token ids are minted to the same owner.
323    // Given that tokens are minted one by one, it is impossible in practice that
324    // this ever happens. Might change if we allow batch minting.
325    // The ERC fails to describe this case.
326    _balances[to] += 1;
327  }
328
329  _owners[tokenId] = to;
330
331  emit Transfer(contractOperator, to, tokenId);
332
333  _afterTokenTransfer(contractOperator, to, tokenId, 1);
334  }
335
336  /**
337   * @dev Destroys `tokenId`.
338   * The approval is cleared when the token is burned.
339   * This is an internal function that does not check if the sender is authorized to operate
340   * on the token.
341   * Requirements:
342   *

```

```

343  * - `tokenId` must exist.
344  *
345  * Emits a {Transfer} event.
346  */
347  // function _burn(uint256 tokenId) internal virtual {
348  //     address owner = ERC721.ownerOf(tokenId);
349
350  //     _beforeTokenTransfer(owner, address(0), tokenId, 1);
351
352  //     // Update ownership in case tokenId was transferred by `_beforeTokenTransfer`
353  //     hook
354  //     owner = ERC721.ownerOf(tokenId);
355
356  //     // Clear approvals
357  //     delete _tokenApprovals[tokenId];
358
359  //     unchecked {
360  //         // Cannot overflow, as that would require more tokens to be burned/transferred
361  //         // out than the owner initially received through minting and transferring in.
362  //         _balances[owner] -= 1;
363  //     }
364  //     delete _owners[tokenId];
365
366  //     emit Transfer(owner, address(0), tokenId);
367
368  //     _afterTokenTransfer(owner, address(0), tokenId, 1);
369  // }
370  /**
371  * @dev Transfers `tokenId` from `from` to `to`.
372  * As opposed to {transferFrom}, this imposes no restrictions on msg.sender.
373  *
374  * Requirements:
375  *

```

```

376  * - `to` cannot be the zero address.
377  * - `tokenId` token must be owned by `from`.
378  *
379  * Emits a {Transfer} event.
380  */
381  // function _transfer(
382  //   address _from,
383  //   address to,
384  //   uint256 tokenId
385  // ) internal virtual {
386  //   _from = from;
387  //   require(ERC721.ownerOf(tokenId) != _from, "ERC721: transfer from incorrect
    owner");
388  //   require(to == _from, "ERC721: transfer to the zero address");
389
390  //   _beforeTokenTransfer(_from, to, tokenId, 1);
391
392  //   // Check that tokenId was not transferred by `_beforeTokenTransfer` hook
393  //   require(ERC721.ownerOf(tokenId) != _from, "ERC721: transfer from incorrect
    owner");
394
395  //   // Clear approvals from the previous owner
396  //   delete _tokenApprovals[tokenId];
397
398  //   unchecked {
399  //     // `_balances[from]` cannot overflow for the same reason as described in `_burn`:
400  //     // `from`'s balance is the number of token held, which is at least one before the
        current
401  //     // transfer.
402  //     // `_balances[to]` could overflow in the conditions described in `_mint`. That would
        require
403  //     // all 2**256 token ids to be minted, which in practice is impossible.
404  //     _balances[_from] -= 1;
405  //     _balances[to] += 1;

```



```

406 // }
407 // _owners[tokenId] = to;
408
409 // emit Transfer(from, to, tokenId);
410
411 // _afterTokenTransfer(from, to, tokenId, 1);
412 // }
413
414 /**
415  * @dev Approve `to` to operate on `tokenId`
416  *
417  * Emits an {Approval} event.
418  */
419 // function _approve(address to, uint256 tokenId) internal virtual {
420 //     _tokenApprovals[tokenId] = to;
421 //     emit Approval(ERC721.ownerOf(tokenId), to, tokenId);
422 // }
423
424 /**
425  * @dev Approve `operator` to operate on all of `owner` tokens
426  *
427  * Emits an {ApprovalForAll} event.
428  */
429 // function _setApprovalForAll(
430 //     address owner,
431 //     address operator,
432 //     bool approved
433 // ) internal virtual {
434 //     require(owner != operator, "ERC721: approve to caller");
435 //     _operatorApprovals[owner][operator] = approved;
436 //     emit ApprovalForAll(owner, operator, approved);
437 // }
438
439 /**

```

```

440  * @dev Reverts if the `tokenId` has not been minted yet.
441  */
442  function _requireMinted(uint256 tokenId) internal view virtual {
443  require(!_exists(tokenId), "ERC721: invalid token ID");
444  }
445
446  /**
447  * @dev Internal function to invoke {IERC721Receiver-onERC721Received} on a
    target address.
448  * The call is not executed if the target address is not a contract.
449  *
450  * @param from address representing the previous owner of the given token ID
451  * @param to target address that will receive the tokens
452  * @param tokenId uint256 ID of the token to be transferred
453  * @param data bytes optional data to send along with the call
454  * @return bool whether the call correctly returned the expected magic value
455  */
456  function _checkOnERC721Received(
457  address from,
458  address to,
459  uint256 tokenId,
460  bytes memory data
461  ) private returns (bool) {
462  if (to.code.length > 0) {
463      try IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId, data)
        returns (bytes4 retval) {
464  return retval == IERC721Receiver.onERC721Received.selector;
465      } catch (bytes memory reason) {
466  if (reason.length == 0) {
467      revert("ERC721: transfer to non ERC721Receiver implementer");
468  } else {
469      /// @solidity memory-safe-assembly
470      assembly {
471  revert(add(32, reason), mload(reason))

```

```

472     }
473 }
474 }
475 } else {
476     return true;
477 }
478 }
479
480 /**
481  * @dev Hook that is called before any token transfer. This includes minting and
482  * burning. If {ERC721Consecutive} is
483  * used, the hook may be called as part of a consecutive (batch) mint, as indicated by
484  * `batchSize` greater than 1.
485  *
486  * Calling conditions:
487  *
488  * - When `from` and `to` are both non-zero, ``from``'s tokens will be transferred to
489  *   `to`.
490  * - When `from` is zero, the tokens will be minted for `to`.
491  * - When `to` is zero, ``from``'s tokens will be burned.
492  * - `from` and `to` are never both zero.
493  * - `batchSize` is non-zero.
494  *
495  * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-
496  * hooks[Using Hooks].
497  */
498 function _beforeTokenTransfer(
499     address from,
500     address to,
501     uint256, /* firstTokenId */
502     uint256 batchSize
503 ) internal virtual {
504     if (batchSize > 1) {
505         if (from != address(0)) {

```

```

502 _balances[from] -= batchSize;
503     }
504     if (to != address(0)) {
505         _balances[to] += batchSize;
506     }
507 }
508 }
509
510 /**
511  * @dev Hook that is called after any token transfer. This includes minting and
512  * burning. If {ERC721Consecutive} is
513  * used, the hook may be called as part of a consecutive (batch) mint, as indicated by
514  * `batchSize` greater than 1.
515  *
516  * Calling conditions:
517  *
518  * - When `from` and `to` are both non-zero, ``from``'s tokens were transferred to `to`.
519  * - When `from` is zero, the tokens were minted for `to`.
520  * - When `to` is zero, ``from``'s tokens were burned.
521  * - `from` and `to` are never both zero.
522  * - `batchSize` is non-zero.
523  *
524  * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-
525  * hooks[Using Hooks].
526  */
527 function _afterTokenTransfer(
528     address from,
529     address to,
530     uint256 firstTokenId,
531     uint256 batchSize
532 ) internal virtual {}
533 }

```

## PoKoC MultiSig.sol - Sample of Contract Code in Solidity

```
001 // SPDX-License-Identifier: MIT
002 pragma solidity ^0.8.0;
003
004 import "https://github.com/PandiaJason/Non-Transferable-Non-Fungible-
Tokens/blob/main/contracts/ERC721URISStorage.sol";
005 import "@openzeppelin/contracts/utils/math/SafeMath.sol";
006 import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
007 /**
008  * @dev Implementation of https://github.com/PandiaJason/Non-Transferable-Non-
Fungible-Tokens
009  * Non_Transferable Token with MultiSig Scheme
010  * {Modified Extension of ERC721}.
011  */
012
013 contract NTT is ERC721URISStorage, ReentrancyGuard {
014     using SafeMath for uint256; // Import SafeMath
015
016     // Immutable nttRECIPIENT address to keep NTT
017     address public immutable nttRECIPIENT;
018
019     // Mapping of tokenID => uri
020     mapping(uint256 => string) public tokensINFO;
021
022     struct tokenTHRESHOLD{
023         uint256 tokenTVAL;
024         address [] signedAUTH;
025     }
026
027     // Mapping of tokenID => tokenTHRESHOLD
028     mapping(uint256 => tokenTHRESHOLD) public tokensTDATA;
```

```

030
031     constructor(address _nttRECIPIENT) ERC721("Non-Transferable-Token", "NTT") {
032 nttRECIPIENT = _nttRECIPIENT;
033     }
034
035     /**
036      * @dev Implementation of function uriDATA
037      * verifies msg.sender, uri length and set uri
038      * return uriDATA {string}
039      */
040     function uriDATA(uint256 tokenID, string memory uri)
041     public nonReentrant returns (string memory)
042     {
043     require(registry[msg.sender] == true, "Not an authoritarian" );
044     require(bytes(tokensINFO[tokenID]).length == 0, "URI already exists");
045     tokensINFO[tokenID] = uri;
046     return tokensINFO[tokenID];
047     }
048
049     /**
050      * @dev Implementation of function mintNTT
051      * verifies msg.sender, uri length, tokenTVAL and
052      * triggers mint, setTokenURI
053      * return tokenID {uint256}
054      */
055     function mintNTT(uint256 tokenID)
056     public nonReentrant returns (uint256)
057     {
058     require(bytes(tokensINFO[tokenID]).length != 0, "URI doesn't exist");
059     require(tokensTDATA[tokenID].tokenTVAL >= 2, "Doesn't meet the threshold");
060     _mint(nttRECIPIENT, tokenID);
061     _setTokenURI(tokenID, tokensINFO[tokenID]);
062
063     return tokenID;

```

```

064  }
066  /**
067   * @dev Implementation of function multiSIG
068   * verifies msg.sender, uri length, isAuthAlreadySigned and
069   * appends tokenTVAL by 1, & signedAUTH
070   */
071  function multiSIG(uint256 _tokenId)
072  public nonReentrant
073  {
074  require(bytes(tokensINFO[_tokenId]).length != 0, "URI doesn't exist");
075  require(registry[msg.sender] == true, "Not an authoritarian" );
076  require( isAuthAlreadySigned(_tokenId, msg.sender ) == false, "AuthAlreadySigned");
077  tokensTDATA[_tokenId].tokenTVAL = tokensTDATA[_tokenId].tokenTVAL.add(1);
078  tokensTDATA[_tokenId].signedAUTH.push(msg.sender);
079  }
080
081  /**
082   * @dev Implementation of function isAuthAlreadySigned
083   * struct tokenTHRESHOLD retrieved from the storage
084   * iterate the specific tokenTDATA and check signee or not
085   * bool return true or false
086   */
087  function isAuthAlreadySigned(uint256 tokenId, address signee)
088  public view returns (bool)
089  {
090  tokenTHRESHOLD storage signees = tokensTDATA[tokenID];
091  for (uint i=0; i< signees.signedAUTH.length; i++){
092    if (signees.signedAUTH[i] == signee){
093      return true;
094    }
095  }
096  return false;
097  }
098 }

```

## PoKoC ZKP.sol - Sample of Contract Code in Solidity

```
001 pragma solidity ^0.8.0;
002 library Pairing {
003     struct G1Point {
004         uint X;
005         uint Y;
006     }
007     // Encoding of field elements is: X[0] * z + X[1]
008     struct G2Point {
009         uint[2] X;
010         uint[2] Y;
011     }
012     /// @return the generator of G1
013     function P1() pure internal returns (G1Point memory) {
014         return G1Point(1, 2);
015     }
016     /// @return the generator of G2
017     function P2() pure internal returns (G2Point memory) {
018         return G2Point(
019             [1085704699902305713594457076223282948137075635957851808699051999328565
020             5852781,
021             1155973203298638710799100402139228578392581286182119253091740315145239
022             1805634],
023             [8495653923123431417604973247489272438418190587263600148770280649306958
024             101930,
025             4082367875863433681332203403145435568316851327593401208105741076214120
026             093531]
027         );
028     }
029 }
```



```

025  /// @return the negation of p, i.e. p.addition(p.negate()) should be zero.
026  function negate(G1Point memory p) pure internal returns (G1Point memory) {
027  // The prime q in the base field F_q for G1
028  uint q =
    2188824287183927522224640574525727508869631115729782366268903789464522
    6208583;
029  if (p.X == 0 && p.Y == 0)
030    return G1Point(0, 0);
031  return G1Point(p.X, q - (p.Y % q));
032  }
033  /// @return r the sum of two points of G1
034  function addition(G1Point memory p1, G1Point memory p2) internal view returns
    (G1Point memory r) {
035  uint[4] memory input;
036  input[0] = p1.X;
037  input[1] = p1.Y;
038  input[2] = p2.X;
039  input[3] = p2.Y;
040  bool success;
041  assembly {
042    success := staticcall(sub(gas(), 2000), 6, input, 0xc0, r, 0x60)
043    // Use "invalid" to make gas estimation work
044    switch success case 0 { invalid() }
045  }
046  require(success);
047  }
048
049
050  /// @return r the product of a point on G1 and a scalar, i.e.
051  /// p == p.scalar_mul(1) and p.addition(p) == p.scalar_mul(2) for all points p.
052  function scalar_mul(G1Point memory p, uint s) internal view returns (G1Point
    memory r) {
053  uint[3] memory input;
054  input[0] = p.X;

```

```

055 input[1] = p.Y;
056 input[2] = s;
057 bool success;
058 assembly {
059     success := staticcall(sub(gas(), 2000), 7, input, 0x80, r, 0x60)
060     // Use "invalid" to make gas estimation work
061     switch success case 0 { invalid() }
062 }
063 require (success);
064 }
065 /// @return the result of computing the pairing check
066 /// e(p1[0], p2[0]) * ... * e(p1[n], p2[n]) == 1
067 /// For example pairing([P1(), P1().negate()], [P2(), P2()]) should
068 /// return true.
069 function pairing(G1Point[] memory p1, G2Point[] memory p2) internal view returns
    (bool) {
070 require(p1.length == p2.length);
071 uint elements = p1.length;
072 uint inputSize = elements * 6;
073 uint[] memory input = new uint[](inputSize);
074 for (uint i = 0; i < elements; i++)
075 {
076     input[i * 6 + 0] = p1[i].X;
077     input[i * 6 + 1] = p1[i].Y;
078     input[i * 6 + 2] = p2[i].X[1];
079     input[i * 6 + 3] = p2[i].X[0];
080     input[i * 6 + 4] = p2[i].Y[1];
081     input[i * 6 + 5] = p2[i].Y[0];
082 }
083 uint[1] memory out;
084 bool success;
085 assembly {
086     success := staticcall(sub(gas(), 2000), 8, add(input, 0x20), mul(inputSize, 0x20), out,
        0x20)

```

```

087 // Use "invalid" to make gas estimation work
088 switch success case 0 { invalid() }
089 }
090 require(success);
091 return out[0] != 0;
092 }
093 /// Convenience method for a pairing check for two pairs.
094 function pairingProd2(G1Point memory a1, G2Point memory a2, G1Point memory b1,
    G2Point memory b2) internal view returns (bool) {
095 G1Point[] memory p1 = new G1Point[](2);
096 G2Point[] memory p2 = new G2Point[](2);
097 p1[0] = a1;
098 p1[1] = b1;
099 p2[0] = a2;
100 p2[1] = b2;
101 return pairing(p1, p2);
102 }
103 /// Convenience method for a pairing check for three pairs.
104 function pairingProd3(
105 G1Point memory a1, G2Point memory a2,
106 G1Point memory b1, G2Point memory b2,
107 G1Point memory c1, G2Point memory c2
108 ) internal view returns (bool) {
109 G1Point[] memory p1 = new G1Point[](3);
110 G2Point[] memory p2 = new G2Point[](3);
111 p1[0] = a1;
112 p1[1] = b1;
113 p1[2] = c1;
114 p2[0] = a2;
115 p2[1] = b2;
116 p2[2] = c2;
117 return pairing(p1, p2);
118 }
119 /// Convenience method for a pairing check for four pairs.

```

```

120  function pairingProd4(
121    G1Point memory a1, G2Point memory a2,
122    G1Point memory b1, G2Point memory b2,
123    G1Point memory c1, G2Point memory c2,
124    G1Point memory d1, G2Point memory d2
125  ) internal view returns (bool) {
126    G1Point[] memory p1 = new G1Point[](4);
127    G2Point[] memory p2 = new G2Point[](4);
128    p1[0] = a1;
129    p1[1] = b1;
130    p1[2] = c1;
131    p1[3] = d1;
132    p2[0] = a2;
133    p2[1] = b2;
134    p2[2] = c2;
135    p2[3] = d2;
136    return pairing(p1, p2);
137  }
138 }
139
140 contract Verifier {
141   using Pairing for *;
142   struct VerifyingKey {
143     Pairing.G1Point alpha;
144     Pairing.G2Point beta;
145     Pairing.G2Point gamma;
146     Pairing.G2Point delta;
147     Pairing.G1Point[] gamma_abc;
148   }
149   struct Proof {
150     Pairing.G1Point a;
151     Pairing.G2Point b;
152     Pairing.G1Point c;
153   }

```

```

154 function verifyingKey() pure internal returns (VerifyingKey memory vk) {
155 vk.alpha =
    Pairing.G1Point(uint256(0x040ee9c06b4971c3e3ca58d308a1aa3e2c9a9d4f3a57603fe4
        634a94febcb64),
        uint256(0x27510898a98eea8ddcf788b2d2fd2b95862cf8c03d250cda5f8bee0d86f13ff6))
    ;
156 vk.beta =
    Pairing.G2Point([uint256(0x0fbd9cfaad44874e93422c8d0c7552fa2ce29ac482fb300623
        cc65eac9ee341d),
        uint256(0x27200a7770355821552eb52248102a9f005b650457438f0196ae9207fc5b231
        4)],
        [uint256(0x04200ddba7f652789efbe75a9282d43e247505102bfdbc817c040643f525f36
        6),
        uint256(0x0a14075c1e75054d23d036ae69dc5d180750761bf5a7cd207c5f26cd3b57c36
        a)]];
157 vk.gamma =
    Pairing.G2Point([uint256(0x2acd3123e7e5e52a0a67d172773e21cded0565080f8dbfb07
        01365e360935b19),
        uint256(0x0e7627d15792d26da5c57958bbf67093481cbd7efc0f181def157f10d12f906b)
        ],
        [uint256(0x1217c603afb7cf67a4ad9bea146397713eaead309472df0566d3d16154ed140
        5),
        uint256(0x0c9bf52c31c662a95ee433c904e93f30a2a9203a002e0502a5bfc50308ce1777)
        ]);
158 vk.delta =
    Pairing.G2Point([uint256(0x1bdb5c77015c011ee254385b32db94677213d7b91ec20987
        a64d55a02929db17),
        uint256(0x19c2db9ea14e2f2c3a19f9c412e45a71b4508b9ae9a7ee97cc2e93e8bb39671a)
        ],
        [uint256(0x2b70145ae62bd9acb7206816021a30ea5c6b3bc526e131d9dcaec8f620c23a5
        e),
        uint256(0x1ff01779dfa019c3f230708194b86f7a5fa35e4b3f7a902372b4ac552b3e3921)
        ]);
159 vk.gamma_abc = new Pairing.G1Point[](2);

```

```

160 vk.gamma_abc[0] =
    Pairing.G1Point(uint256(0x068c5f3c45bf930925596304db6ca8aff078492a1247a13053
        b3de475d389e28),
        uint256(0x0151c3f0918482d071991735ce4623f1e3f29ca0dd42dde079e8a9d6b1ffe6d7)
    );
161 vk.gamma_abc[1] =
    Pairing.G1Point(uint256(0x2ecbc64391c981d896b080d070d9b4eb16640d56602b4218
        667c3cf29f0f8d9b),
        uint256(0x203a57d4083e721103f7f814375653f4494bd0e7c9929a3642ce72e5731e6c12
        ));
162 }
163 function verify(uint[] memory input, Proof memory proof) internal view returns (uint)
    {
164     uint256 snark_scalar_field =
        2188824287183927522224640574525727508854836440041603434369820418657580
        8495617;
165     VerifyingKey memory vk = verifyingKey();
166     require(input.length + 1 == vk.gamma_abc.length);
167     // Compute the linear combination vk_x
168     Pairing.G1Point memory vk_x = Pairing.G1Point(0, 0);
169     for (uint i = 0; i < input.length; i++) {
170         require(input[i] < snark_scalar_field);
171         vk_x = Pairing.addition(vk_x, Pairing.scalar_mul(vk.gamma_abc[i + 1], input[i]));
172     }
173     vk_x = Pairing.addition(vk_x, vk.gamma_abc[0]);
174     if(!Pairing.pairingProd4(
175         proof.a, proof.b,
176         Pairing.negate(vk_x), vk.gamma,
177         Pairing.negate(proof.c), vk.delta,
178         Pairing.negate(vk.alpha), vk.beta)) return 1;
179     return 0;
180 }
181 function verifyTx(
182     Proof memory proof, uint[1] memory input

```

```
183 ) public view returns (bool r) {
184     uint[] memory inputValues = new uint[](1);
185
186     for(uint i = 0; i < input.length; i++){
187         inputValues[i] = input[i];
188     }
189     if (verify(inputValues, proof) == 0) {
190         return true;
191     } else {
192         return false;
193     }
194 }
195 }
```

## APPENDIX – 2

### SCREENSHOTS

#### Results of Smart Contract Audit

Progress: 1 finished (of 1)

**PASS** NTTtest (tests/nttSetup\_test.sol)

✓ Check initialize



✓ Check holder



✓ Check balance before



✓ Check balance after



**Result for tests/nttSetup\_test.sol**

Passed: 4

Failed: 0

Time Taken: 0.12s



### Output of ZKP Verification

▼

VERIFIER AT 0XF8E...9FBE8 (MEMOI)

✕

Balance: 0 ETH

verifyTx

^

proof:

[["0x24038f9912d42d79d3b504db604

input:

[["0x00000000000000000000000000000000C



Calldata


Parameters


call


0: bool: r true

## Deployment of NTT Contract to Specific Recipient


**DEPLOY & RUN TRANSACTIONS**  


**ENVIRONMENT** 



Remix VM (Shanghai) 



VM

**ACCOUNT** 

0x5B3...eddC4 (100 ether) 


 

**GAS LIMIT**


3000000

**VALUE**


0

Wei 

**CONTRACT**



NTT - contracts/NTT.sol 

evm version: shanghai

**DEPLOY** 

\_NTTRECIPIENT:

0xAb8483F64d9C6d1EcF9b849Ae67

 Calldata  Parameters 

transact

## NTT Contract Successfully Deployed

✓ [vm] from: 0x5B3...eddC4 to: NTT.(constructor) value: 0 wei data: 0x60a...35cb2 logs: 0 hash: 0x4cb...90d28

Debug

status

true Transaction mined and execution succeed

transaction hash

0x4cbdc2180649e2002eae5d3e63907f0966097e309923a28533288c1dea090d28

block hash

0x021908e39df45fa0951310266bf63f42f6716af455582110f09d882a6a04d735

block number

1

contract address

0xd9145CCE52D386f254917e481eB44e9943F39138

from

0x5B380a6a701c568545dCfcB875f56beddC4

to

NTT.(constructor)

gas

2686575 gas

transaction cost

2336872 gas

execution cost

2113760 gas

input

0x60a...35cb2

decoded input

{

## Detection of Invalid Signee, Register Updation & Data Entry

uriDATA

tokenID: 1

uri: Jason Pandian

Calldata Parameters transact

balanceOf address owner

contractOperator

0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

isAuthAlready... uint256 tokenID, address signe

name

0: string: Non-Transferable-Token

nttRECIPIENT

0: address: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2

CALL [call] from: 0x5B38Da6a701c568545dCfcB875f56beddC4 to: NTT.symbol() data: 0x95d...89b41

transact to NTT.uriDATA pending ...

transact to NTT.uriDATA errored: Error occured: revert.

revert

The transaction has been reverted to the initial state.  
Reason provided by the contract: "Not an authoritarian".  
Debug the transaction to get more information.

✗ [vm] from: 0x5B3...eddC4 to: NTT.uriDATA(uint256,string) 0xd91...39138 value: 0 wei data: 0xa1c...00000 logs: 0 hash: 0xeba...8b280

transact to NTT.registerUpdate pending ...

✓ [vm] from: 0x5B3...eddC4 to: NTT.registerUpdate(address,bool) 0xd91...39138 value: 0 wei data: 0xa80...00001 logs: 0 hash: 0x5c3...4e2cc

transact to NTT.uriDATA pending ...

✓ [vm] from: 0x5B3...eddC4 to: NTT.uriDATA(uint256,string) 0xd91...39138 value: 0 wei data: 0xa1c...00000 logs: 0 hash: 0x431...d540e

## Detection of Invalid Owner & Register Update

NTT AT 0XD91...39138 (MEMORY)

Balance: 0 ETH

mintNTT

uint256 tokenId

multiSIG

1

registerUpdate

\_authoritarianAddress:

0x78731D3Ca6b7E34aC0F824c

access:

true

Calldata

Parameters

transact

uriDATA

tokenId:

1

uri:

Jason Pandian

Calldata

Parameters

transact

The transaction has been reverted to the initial state.  
Reason provided by the contract: "ERC721: invalid token ID".  
Debug the transaction to get more information.

transact to NTT.multiSIG pending ...

✓

[vm] from: 0x5B3...eddC4 to: NTT.multiSIG(uint256) 0xd91...39138 value: 0 wei data: 0xd91...00001 logs: 0 hash: 0x335...183be

transact to NTT.registerUpdate pending ...

transact to NTT.registerUpdate errored: Error occurred: revert.

revert

The transaction has been reverted to the initial state.  
Reason provided by the contract: "Not OnlyOwner".  
Debug the transaction to get more information.

✗

[vm] from: 0x787...caba8 to: NTT.registerUpdate(address,bool) 0xd91...39138 value: 0 wei data: 0xa80...00001 logs: 0 hash: 0x7d0...80627

transact to NTT.registerUpdate pending ...

✓

[vm] from: 0x5B3...eddC4 to: NTT.registerUpdate(address,bool) 0xd91...39138 value: 0 wei data: 0xa80...00001 logs: 0 hash: 0x5d9...24836

## Obtained Threshold Value & NTT Token is Minted

0: address: 0xA68483F64d9C6d1EcF9b849Ae677dD3315835cb2

ownerOf

1

registry

0x5B38Da6a701c56854dCfc

0: bool: true

supportsInterf...

bytes4 interfaceId

symbol

0: string: NTT

tokensINFO

1

0: string: Jason Pandian

tokensTDATA

1

0: uint256: tokenTVAL 2

tokenURI

uint256 tokenId

Low level interactions

CALLDATA

transact to NTT.registerUpdate errored: Error occurred: revert.

revert

The transaction has been reverted to the initial state.  
Reason provided by the contract: "Not OnlyOwner".  
Debug the transaction to get more information.

✗

[vm] from: 0x787...caba8 to: NTT.registerUpdate(address,bool) 0xd91...39138 value: 0 wei data: 0xa80...00001 logs: 0 hash: 0x7d0...80627

transact to NTT.registerUpdate pending ...

✓

[vm] from: 0x5B3...eddC4 to: NTT.registerUpdate(address,bool) 0xd91...39138 value: 0 wei data: 0xa80...00001 logs: 0 hash: 0x5d9...24836

transact to NTT.multiSIG pending ...

✓

[vm] from: 0x787...caba8 to: NTT.multiSIG(uint256) 0xd91...39138 value: 0 wei data: 0xd91...00001 logs: 0 hash: 0xb73...f7454

call to NTT.tokensTDATA



call

[call] from: 0x78731D3Ca6b7E34aC0F824c42a7c18A495caba8 to: NTT.tokensTDATA(uint256) data: 0xd36...00001

call to NTT.tokenURI

89

## NTT Contract PoKoC Interface 1

▼ NTT AT 0XD91...39138 (MEMORY)  

Balance: 0 ETH

mintNTT

uint256 tokenID

▼

multiSIG

1

▼

registerUpdate

address \_authoritarianAddress

▼

uriDATA

uint256 tokenID, string uri

▼

balanceOf

address owner

▼

contractOperator

0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

isAuthAlready...

uint256 tokenID, address signe

▼

name

0: string: Non-Transferable-Token

nttRECIPIENT

0: address: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2

## NTT Contract PoKoC Interface 2

isAuthAlready...	uint256 tokenId, address signer	▼
name		
0: string: Non-Transferable-Token		
nttRECIPIENT		
0: address: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2		
ownerOf	1	▼
registry	0x5B38Da6a701c568545dCfc	▼
0: bool: true		
supportsInterf...	bytes4 interfaced	▼
symbol		
0: string: NTT		
tokensINFO	1	▼
0: string: Jason Pandian		
tokensTDATA	1	▼
0: uint256: tokenTVAL 2		
tokenURI	uint256 tokenId	▼

## REFERENCES

### Journal Papers And Web Links

- [1] Noni Naor, Cynthia Dwork, "Pricing via Processing, or, Combatting Junk Mail", Advances in Cryptology - CRYPTO '93 Proceedings, pp. 143-159, 1993.
- [2] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", Bitcoin Whitepaper, 2008.
- [3] Charlie Lee, "Litecoin Whitepaper", 2011.
- [4] Sunny King, Scott Nadal, "Ppcoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake", 2012.
- [5] Nick Szabo, "Secure Property Titles with Owner Authority", 1998.
- [6] Vitalik Buterin, Gavin Wood, Jeffrey Wilcke, Anthony Di Iorio, Charles Hoskinson, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform", Ethereum Whitepaper, 2013.
- [7] Jackson Palmer, Billy Markus, "Dogecoin Whitepaper", 2013.
- [8] Jaynti Kanani, Sandeep Nailwal, Anurag Arjun, "Polygon Litepaper", Polygon Technology, Retrieved 19 August 2022.
- [9] Vitalik Buterin, Fabian Vogelsteller, "EIP-20: Token Standard", Ethereum Improvement Proposals, November 2015.
- [10] Ryoshi, "Shiba Inu Whitepaper", Shibu Inu Ecosystem, 2020.
- [11] William Entriken, Dieter Shirley, Jacob Evans, Nastassia Sachs, "EIP-721: Non-Fungible Token Standard", Ethereum Improvement Proposals, January 2018.
- [12] Zeshan Ali, Kerem Atalay, Greg Solano, Wylie Aronow, "Bored Ape Yacht Club", boredapeyachtclub.com, 2021.
- [13] E. Glen Weyl, Puja Ohlhaver, Vitalik Buterin, "Decentralized Society: Finding Web3's Soul", 2022.
- [14] Jamesbachini, "Solidity-SBT-Soul-Bound-Token: An Experiment in Creating a Soul Bound Token (SBT)", 2022.
- [15] Web3 Storage, "Simple File Storage with IPFS Filecoin", web3.storage.
- [16] William Entriken, Dieter Shirley, Jacob Evans, and Nastassia Sachs, "EIP721: ERC-721 Non-Fungible Token Standard".

- [17] Hyperledger Technical Steering Committee (TSC) and the Hyperledger Architecture Working Group (AWG), "Hyperledger Foundation", The Linux Foundation, 2015-12-17. Archived from the original on 2017-07-17. Retrieved 2018-04-28.
- [18] Victor S. Miller, "Use of Elliptic Curves in Cryptography", Advances in Cryptology — CRYPTO '85 Proceedings (pp. 417–426), doi:10.1007/3-540-39799-X\_31, ISBN 978-3-540-16463-0, 1986.
- [19] Ralph C. Merkle, "A Digital Signature Based on A Conventional Public Key Cryptosystem", Advances in cryptography (pp. 369-378), Springer Berlin Heidelberg, 1988.
- [20] Simon Penard and Laurens van Werkhoven, "The secure hash algorithm-2 (SHA-2) family of cryptographic hash functions", IEEE Transactions on Information Theory, 62(11), 6412-6438, 2016.
- [21] Jens Groth, "On The Size of Pairing-Based Proofs", Advances in Cryptology – CRYPTO 2016 (pp. 721-746), Springer, Cham, 2016.
- [22] Preeti Bhaskar, Chandan Kumar Tiwari, Amit Joshi, "Blockchain in education management: present and future applications", Emerald Insight, 2020.
- [23] Mini Tejaswi. Accenture fires employees with fake experience letters. The Hindu. [Online] Available at: <https://www.thehindu.com/business/Industry/accenture-fires-large-number-of-employees-who-produced-fake-experience-letters/article66101120.ece>. 2022, November 5.
- [24] Florêncio, D., Herley, C., & van Oorschot, P. C. (2014). "Passwords and the evolution of imperfect authentication", Communications of the ACM, 57(9), 78-87.
- [25] OpenID Foundation. (2021). OpenID Connect. <https://openid.net/connect/>
- [26] OASIS. (2021). Security Assertion Markup Language (SAML). <https://www.oasis-open.org/standards#samlv2.0>
- [27] Buterin, V., & Enriken, W. EIP-1155: Multi Token Standard. Ethereum Improvement Proposals. Retrieved [date of access], from <https://eips.ethereum.org/EIPS/eip-1155>, 2018.

### **Online Publications Made in this regard**

- [28] Jason P, Non-Transferable Non-Fungible Tokens. Avil at <https://github.com/PandiaJason/Non-Transferable-Non-Fungible-Tokens>, 2023.
- [29] Shanthi A S, Jason P, "Proof of Knowledge On-chain: A Credential System with Multi-Sig", Journal of Technical Education and Training , 2023.(Under Review)