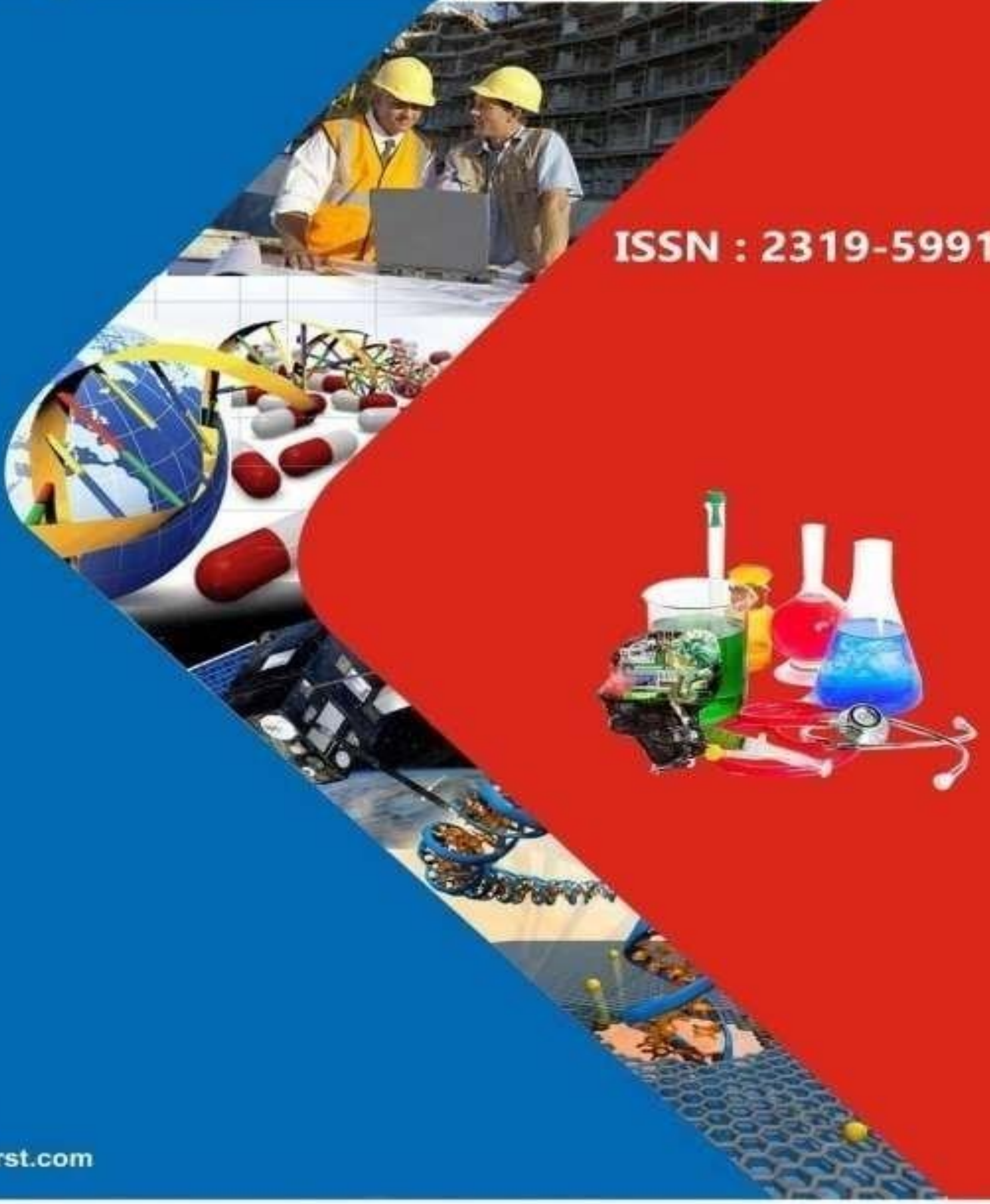
International Journal of Engineering Research and Science & Technology

IJERST



ISSN : 2319-5991

www.ijerst.com

Email: editor@ijerst.com or editor.ijerst@gmail.com

I

ISSN 2319-5991 www.ijerst.com

Vol. 20, Issue 3, 2024 https://doi.org/10.62643/ijerst.2024.v20.i3.pp353-365

"MSC: A Novel Chameleon Hash-Based Off-Chain Storage Framework for Metaverse Applications"

Dr. GANDHAVALLA RAO SAMBASIVARAO, Dr. MOHD ATEEQ AHMED, Dr. FARHEEN SULTANA
Department of Information technology
Nawab Shah Alam Khan College of Engineering and Technology (NSAKCET

ABSTRACT

With its tamper-resistant features and growing reliability, blockchain has become an ideal platform for decentralised applications; nevertheless, it has also introduced new concerns related to on-chain overhead. An intriguing research question on the safe reduction of on-chain storage overhead has emerged in response to the emergence of metaverserelated smart contracts on blockchain. An innovative decentralised system architecture that allows code or stored data changes without altering on-chain data is introduced in this study as the Metaverse Off-chain Storage architecture based on Chameleon hash (MSC). To keep the index consistent even when data is modified, decentralised apps use the Chameleon hash to compute it. With the right authentication procedures in place, data may be stored off-chain at the same time. When compared to other frameworks of its kind, testing findings reveal that MSC uses less on-chain storage. When compared to smart contract data storage directly, MSC also drastically reduced overhead.

Keywords: blockchain | chameleon hash | distributed system | metaverse | storage

1 | Introduction

When we talk about a shared virtual realm that merges physical and digital realities, we're talking about the metaverse. The metaverse makes use of a number of technologies, including blockchain, which is widely utilised for digital asset management. But since blockchain is a distributed ledger technology, metaverse-related decentralised applications (dApps) built on it may grow in size very quickly. Data from the entity blockchain ledger must be saved by every node, which limits the amount of data that can be stored on-chain [1, 2]. An essential part of the metaverse are digital assets based on the blockchain, as seen in Figure 1. Due mostly to data produced by decentralised applications (dApps) linked to the metaverse, the ever-increasing size of the blockchain is imposing limits on the volume of transactions. Consequently, an off-chain storage solution is required for metaverse applications in order to set up decentralised smart contracts and lower the cost of on-chain data storage.

Storage infrastructures based on smart contracts have been suggested; methods for storing key-value data based on smart contracts are provided by Certledger [3], Smart Contract-based PKI (SCPKI) [4], and TrustCA [5]. Having said that, these suggestions are storage inefficient and come with big headers. Their recommendation to reduce on-chain storage costs is to use IPFS for data storage and then store data indices on the blockchain. These suggestions highlight the method's use for the safe upkeep of Public Key Infrastructure (PKI) certificates as altering IPFS data necessitates obtaining a new index.

I

ISSN 2319-5991 www.ijerst.com

Vol. 20, Issue 3, 2024 https://doi.org/10.62643/ijerst.2024.v20.i3.pp353-365

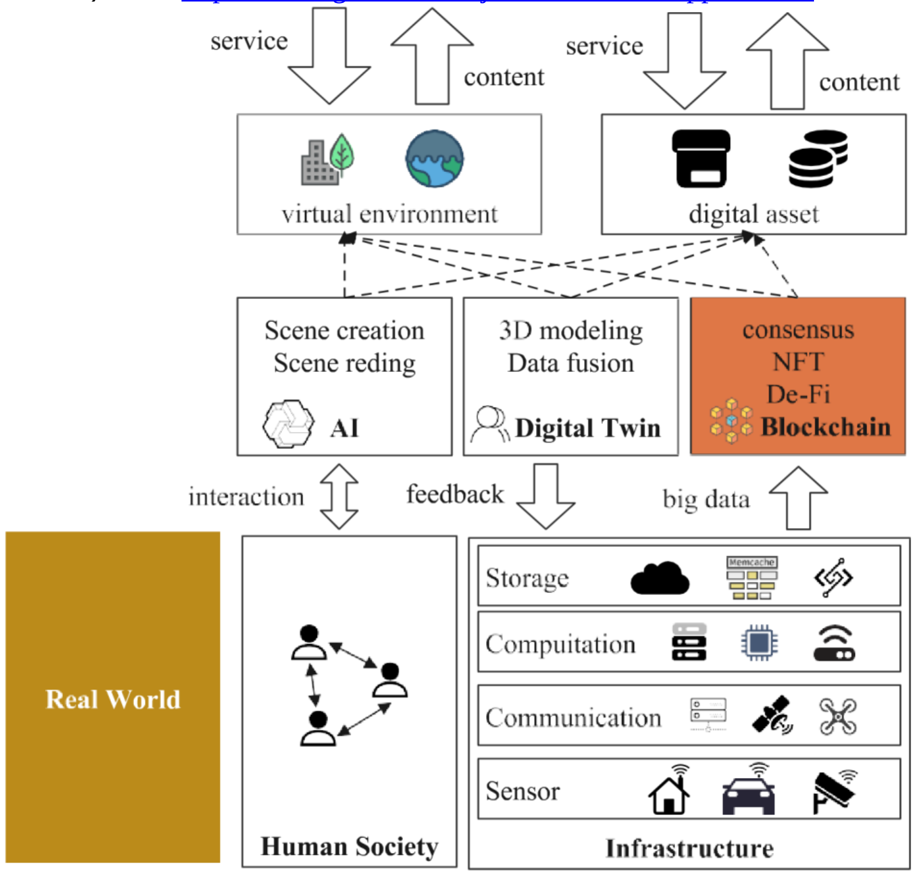


FIGURE 1 | The architecture of metaverse in integration of the MSC.

with limited update needs. In practical applications, data up- dates are a common requirement. For example, domain name administrators need to frequently configure domain resolu- tion information in DNS. This leads to more overhead while sending transactions to call smart contracts after updating the data stored in IPFS. Namecoin [6] introduced a solution involv- ing the creation of a new blockchain dedicated to storing and managing key-value data. Indeed, as mentioned by Sayeed [7], Internet infrastructure built upon existing blockchains can offer enhanced security. Hence, there is a demand for a frame- work to implement off-chain data storage utilizing existing pub- lic blockchains.

This paper introduced a unique framework named Metaverse Off-chain Storage Framework based on Chameleon hash (MSC) for implementing metaverse applications. The difference be- tween MSC and IPFS lies in MSC's capability to alter data in distributed storage without necessitating changes to the index. Data's chameleon hash is used as an index to guarantee that the index remains unaltered following data modifications. In addition, the storage of frequently updated data is made easier

2 | Related Work

Trusted third-party (TTP) solutions in Internet infrastructures have a centralised architectural design, which causes a lot of problems [8]. These designs must have a decentralised architecture in order to be implemented without TTP. Numerous plans to construct blockchain-based Internet infrastructure have been put forth.

There were some suggestions for PKI implementation architectures that relied on blockchain technology. A number of organisations have suggested smart contract-based architectures to incorporate PKI, including CertLedger[3], SCPKI[4], TrustCA[5], and IKP [9]. Certificate revocation and transparency were introduced by CertLedger [3] using blockchain technology. An attribute storage architecture was built by SCPKI [4]. While IPFS stores data in the light version, the smart

ISSN 2319-5991 www.ijerst.com

Vol. 20, Issue 3, 2024 https://doi.org/10.62643/ijerst.2024.v20.i3.pp353-365

contract stores attribute data in the full version. By utilising IPFS, TrustCA [5] has reduced the on-chain storage overhead and implemented smart contract-based certificate transparency. There is a new blockchain-based PKI proposal called IKP [9] that can automatically respond to Certificate Authority (CA) misbehaviour and incentivises individuals who help find it. It uses smart contracts to store PKI-related data and has detectors that watch the smart contract logs for CA misconduct. For the purpose of managing public key infrastructure (PKI), Yakubov [10] suggested a system that uses blockchain technology to issue, validate, and revok certificates. By establishing a trust chain between smart contracts, this framework makes it possible to store certificate data inside them. The use of on-chain storage in these proposals, however, raises the possibility of additional on-chain storage overhead. While some of these suggestions do use IPFS to lessen the burden of on-chain storage, this architecture may make it difficult to make frequent changes to stored data.

The use of blockchain technology for storage has also been considered in other proposals. Storage architectures based on the blockchain have been suggested by Saqib Ali and Shangping Wang. The PingER decentralised data storage and access framework was created by Saqib Ali [11]. It entails putting the file into a P2P network, enclosing it in a transaction, and then sending it to the blockchain. Shangping Wang [12] utilizes Ethereum smart contracts to store encrypted file parameters. The proposed architecture leverages smart contracts as a TTP, allowing users to decrypt data in cloud storage for a legitimate access time. These proposals use blockchain as a TTP to ensure that data cannot be which store the resolution information of domain names in a specific smart contract. While Namecoin and DecDNS proposed new blockchains for data storage, they are susceptible to attacks. ENS [14] stores data in smart contracts, and the on-chain stor- age overhead is very high. Additional overhead is in- curred when there's a need to modify the data.

There are also proposals to use the chameleon hash to design architectures. Song Guo proposed an EC-based chameleon hash scheme, which can derive multiple public keys from private key to ensure the privacy of users [15]. IB-CH [16] is a signature key distribution scheme based on the chameleon hash, which can reduce the computational overhead and the size of public pa- rameters. Songwei Fan proposed an editable blockchain scheme based on the chameleon hash [17]. It uses the chameleon hash to ensure that after modifying block, the overall structure of block- chain is not affected. They have editable architectures designed using chameleon hash, which embodies the main characteris- tics of chameleon hash. However, these architectures have not been used to implement off-chain storage to reduce the cost of on-chain storage. MSC is the first framework implement off- chain storage based on chameleon hash.

3 | Preliminaries

Chameleon hash was introduced in [18], which is the basis of ODSC. It is a hash function with a trapdoor; the private key owner can update the data without causing the output hash value to change. Elliptic curve is used in MSC to construct the chameleon hash function for implementing MSC due to its abil- ity to provide better security with a shorter private key. Abelian group Gp is used to construct chameleon hash, where p represents a large prime number. To ensure security, the cardi- nality of Gp should be divisible by another large prime number q [19]. A point O is chosen from Gp as the base point for calcu- lations. In the MSC-based demo program for testing, the P-256 curve is chosen for implementation, but in practical applications, any legally defined elliptic curve on a finite field can be selected. Initially, $x \in [1, q - 1]$ is selected as the chameleon key, then $y = xO$ is calculated as the public key. In addition, a hash func- tion h is needed to map the original message m1 to a large integer in order to compute the hash of a message m1. When calculating the hash value of message m1 for the first time, a random number r1 must be chosen. Then, the chameleon hash value CH(m1, y, r1) of message m1 can be computed as follows: modified and cannot be used to store modifiable data. $CH(m1, y, r1) = h(m1) \cdot O + r1 \cdot y$ (1) Namecoin [6], DecDNS [13], and Ethereum name service (ENS) [14] are domain name services based on blockchain. Namecoin [6] implemented the storage of key-value data within its blockchain transaction database based on the code of Bitcoin. Namecoin supports ".bit" address, after paying a fixed registration fee, pub- When modifying the data associated with the chameleon hash, assuming the new message is m2, a new random num- ber r2 can be obtained through the collision finding algorithm Find(m1, r1, m2, x) as lic keys can be added to a domain to enable PKI. DecDNS [13] proposes a distributed data storage architecture, which stores the $Find(m1, r1, m2, x) = (xr1 + h(m1) - h(m2))x^{-1}$ (2) verification information of files on blockchain, and distributed nodes store the resolution information of domain names. ENS To ensure that $CH(m1, y, r1) = CH(m2, y, r2)$, expand the formula as is a domain name services based on Ethereum smart contract, $h(m1) \cdot O + xr1 \cdot O = h(m2) \cdot O + xr2 \cdot O$ Hence, the collision algorithm's calculation formula can be de- rived. This approach ensures that the output hash value remains unchanged when updating the data. On the verifier's side, the verifier can calculate the new hash

ISSN 2319-5991 www.ijerst.com

Vol. 20, Issue 3, 2024 https://doi.org/10.62643/ijerst.2024.v20.i3.pp353-365

output to verify whether the random number r2 has been generated correctly with the private key . Only the holder of the private key can generate a valid random number r2, making the chameleon hash unforgeable.

3 | Framework Design

MSC is used to store off-chain data, and it needs to rely on blockchain to store the digest of data to ensure credibility and transparency of data. MSC can be used to implement internet infrastructure such as PKI, BGP, and DNS. Compared with other similar frameworks, MSC can reduce the storage over- head on the chain, which is conducive to store data that need to be changed frequently. In MSC, there are two distinct types of nodes:

- Full Node (*FN*): These nodes run the full node program of blockchain and are capable of broadcasting transactions.
- Monitoring Agent (*MAn*): These nodes run the framework, send transactions to the blockchain, and monitor smart con- tract events through Full Nodes (*FN*). The smart contract is needed to validate the nodes' permission. The purpose of the smart contract and an ERC specification[1] will be explained later. The specific is beneficial for implementing MSC-based applications.

FN has the complete blockchain ledger data, and other nodes can broadcast transactions by connecting to *FN* and monitor events on blockchain. In the case of lower security require- ments, existing APIs can be used instead of running a full node server, such as using Infura's services. *MAn* has the capability to deploy smart contracts and manage its own authority within the same application. A P2P network is established among various *MAs* using Kademlia [20], and all of them maintain a database to manage off-chain data.

As depicted in Figure 2, the overall workflow of MSC consists of four distinct steps as follows. Step. I shows the contract cre- ation and data initialization process. Step. II shows the process of a new node obtaining the connection information from smart contract and joining the P2P network. Step. III shows the pro- cess of storing data in MSCbased program. Step. IV shows the process of obtaining stored data from an MSC-based program.

4.1 | Contract Creation and Data Initialization

In Step. I, as illustrated in Figure 2, a genesis node MA_0 serves as a bootstrap node, which is necessary for setting up the P2P network and creating a smart contract on the blockchain. MA_0 sends a transaction to the blockchain through *FN* for the pur- pose of creating and initializing the smart contract. Smart con- tract can be developed according to the the syntax specified and deployed by default method of blockchain. The data that the smart contract needs to initialize are shown in Table 1.

To restrict the operational permissions of the contract, MA_0 sets the attribute, ensuring that only the can manipu- late the information within the smart contract. The *bootstrap* field in the smart contract is filled by MA_0 with its own P2P connection information. And *genesis* is generated after MA_0 creates genesis data and calculates the chameleon hash of data. The genesis data, denoted as *gd*, are created prior to MA_0 ini- tializing the attributes of the smart contract. In alternative sce- narios, *gd* can be configured as needed. Once MA_0 creates *gd*, the signature is calculated using the chameleon hash, and the signature can be employed for verification and as an index for the data. During the initialization process, MA_0 shares *gd* on the P2P network and writes it to the smart contract.

I

ISSN 2319-5991 www.ijerst.com

Vol. 20, Issue 3, 2024 https://doi.org/10.62643/ijerst.2024.v20.i3.pp353-365

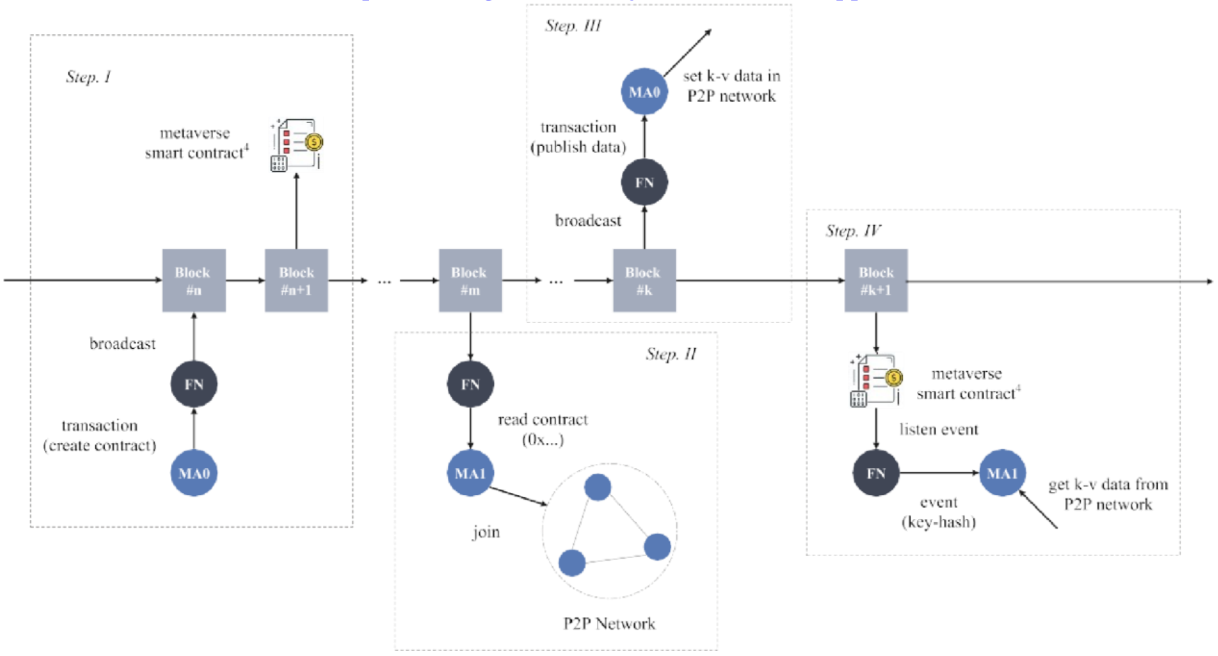


FIGURE 2 | MSC overall workflow. It shows the workflow of MSC, which is divided into four parts.

The pseudo-code of the algorithm responsible for creating the genesis data is shown in Figure 3. To ensure security, it is imperative to specify the version and the latest block hash of the blockchain when the data are created and modified. A list is contained in *gd* to control data write permissions.

The distributed network running this framework will synchro- nize data on the network according to *list*. To support signatures of larger data, the function *Signature* computes a digest of data to shorten the input. According to the pseudocode of Figure 3, the required attributes of genesis data *gd* are shown in Table 2. The *list* in *gd* can be modified by MA_0 without affecting *gd. index* because the index of *gd* can be obtained through the chame- leon hash.

4.2 | Read Attribute and Connect P2P Network

The connection information bootstrap for the genesis node MA_0 is stored in the smart contract. New node can discover other nodes and join the P2P network based on the bootstrap informa- tion. As the Step. II shown in Figure 2, new nodes MA_1 and MA_2 can join the P2P network by the following process.

1. MA_1 reads bootstrap from smart contract, then connects to $MA0$;
2. MA_1 discover other nodes in the P2P network through MA_0 and built its own routing table. Now other nodes can also discover other nodes through MA_1;
3. If another new node, like MA_2, intends to join the P2P net- work at a later stage, it can establish a connection with MA_1 to discover other nodes within the P2P network.

I

ISSN 2319-5991 www.ijerst.com

Vol. 20, Issue 3, 2024 https://doi.org/10.62643/ijerst.2024.v20.i3.pp353-365

TABLE 1 | Smart contract initialization attributes.

Attribute	Description
owner	Owner of smart contract bootstrap
genesis	Index of genesis data on P2P network

correspond to the data lifecycle: *created*, *modified*, and *revoked*. While an event is emitted, the corresponding callback function will be executed. As a result, new nodes can analyze the event log to synchronize data by utilizing these events. New nodes can sequentially traverse the event log in the smart contract to

TABLE 2 | Genesis data initialization attributes.

Attribute	Type	Description
index	string	The chameleon output of genesis data
pubKey	string	Chameleon public key
random	string	Chameleon random number
blockHash	string	Latest block hash
timestamp	int	Data's create timestamp
version	int	Data modification version
list	[]byte	Permission list in the application

5 | Security Analysis

5.1 | Replay Attack

Suppose an adversary wants to use the previous signa- a new

data tuple *data*2 = (*m*2, *pk*, *r*2, *sign*2). Within a data tuple,

m = (*value*|*blockHash*|*version*), nodes in MSC-based application can utilize the *blockHash* and *version* to verify that the data are the latest. may attempt to exploit the original data $data_1$ to overwrite the new data $data_2$, potentially executing a replay attack.

Definition 1. Replay-resistant We say that the MSC-based application is replay-resistant if the success probability of any polynomial-time adversary is negligible in the following experiment:

- A oracle initializes with a chameleon hash key pair (*pk*, *sk*);
- access with message to obtain (*r* , *sign*), where

ISSN 2319-5991 www.ijerst.com

Vol. 20, Issue 3, 2024 https://doi.org/10.62643/ijerst.2024.v20.i3.pp353-365

ture to overwrite the new data. Assume that the original data tuple is *data*1 = (*m*1, *pk*, *r*1, *sign*1), and its owner modifies it to

TABLE 3 | Required attributes of data.

Attribute	Type	Description
index	string	The chameleon output of data
pubKey	string	Chameleon public key
random	string	Chameleon random number
blockHash	string	Latest block hash
timestamp	int	Timestamp when create
version	int	Data modification version
data	[]byte	Byte array of data

*sign*1 = *CH*(*m*, *pk*, *r*1), and *r*1 is randomly chosen by oracle ;

- modifies *blockHash*, *version* or *value* in to construct a new message m^f in polynomial time;
- access oracle with (m^f, *r*1, *sign*1);

The adversary is succeed iff *CH*(*m*, *pk*, *r*1) = *CH*(m^f, *pk*, *r*1).

Proof. According to the security proof of chamelon hash [18], the chameleon hash is collision-resistant. Therefore, no polynomial-time adversary can construct a new message m^f such that *CH*(*m*, *pk*, *r*1) = *CH*(m^f, *pk*, *r*1). □ □

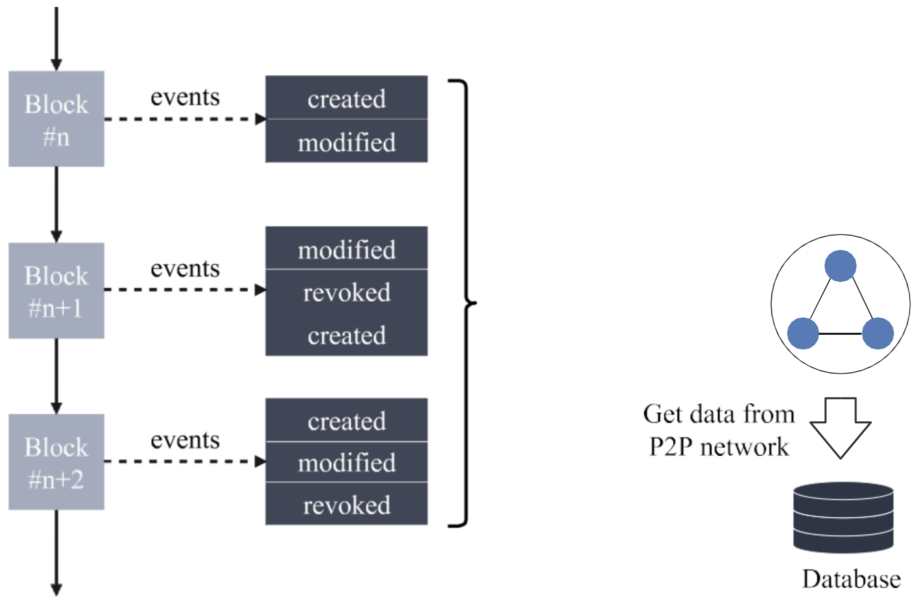


FIGURE 4 | Synchronize workflow.

I

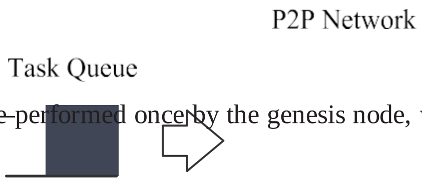
ISSN 2319-5991 www.ijerst.com

Vol. 20, Issue 3, 2024 https://doi.org/10.62643/ijerst.2024.v20.i3.pp353-365

5.2 | Data Security

There are five operations in MSC: *genesis*, *create*, *modify*, *revoke*, and *synchronize*. *genesis* is a unique operation used to initialize MSC, which is similar to *create*. In order to demonstrate the se- curity of these operations, six methods are defined as Table 4. These methods can constitute the five data operations de- scribed above.

public–private key pair is (pk_0, sk_0), and the permission list con- tained in genesis data is *pl*. When initializing MSC, MA_0 con- structs genesis data gd = (pk_0, *timestamp*, *version* = 0, *pl*), then signs *gd* with chameleon hash to get index and random number by *index*, *r* = *MA*0*Sign*(*sk*0, *gd*). *MA*0 constructs genesis message
gm = (*gd*, *index*, *random*). The operation of MA_0 at the time of genesis can be represented as



| Initialization Security

As mentioned above, genesis is the operation that can only be performed once by the genesis node, which corresponds to Step. I of Figure 2. Assuming that the genesis node is MA_0, its

With the operation (5), *index* and *random* are filled in smart contract by genesis node MA_0. The genesis node MA_0 publishes data *gm* to the P2P network with *index*, and emits *created* event in smart contract with parameter *index* and *random*. According to the characteristics of the chameleon hash, only the genesis **TABLE 4** | Methods included in data operations.

Method	Description
MAnSign	Sign data by chameleon hash
MAnResign	Resign data to keep index not change by chameleon hash
Publish	Publish data in P2P network
Emit	Emit event in smart contract
Sync	Synchronize data from P2P network with data index
Verify	Verify data ownership

node MA_0 can generate valid *index* and *random* [18]. In addition, smart contract log is public, and blockchain transactions con- tain sender's information.

Other nodes can use the information and smart contract log to verify whether the data on P2P network have been falsified. Therefore, the unforgeability of chameleon hash and the trans- parency of blockchain ensure that the genesis message *gm* is unalterable.

5.2.1 | Manipulate Data Security

Manipulating data operation is shown as the Step. III of Figure 2. assuming that another node MA_1 needs to cre- ate new data $data_0$ in MSC. MA_1 has public–private key pair (*pk*1, *sk*1). Similar to *genesis*, *MA*1 needs to construct a message d_0 = ($data_0$, *timestamp*, *version* = 0), the signs

MA_1 can send transactions with own address, so only MA_1 can revoke its data. The security of revoke is guaranteed by the char- acteristics of blockchain.

ISSN 2319-5991 www.ijerst.com

Vol. 20, Issue 3, 2024 https://doi.org/10.62643/ijerst.2024.v20.i3.pp353-365

5.2.2 | Synchronize Security

synchronize is the operation of other nodes after monitoring smart contract events. It corresponds to Step. IV of Figure 2. The process has been described in Section 4, and it is the process of other nodes after one node in P2P network performed (6), (7), or (8). We define $Typeevent \in \{created, modified, revoked\}$, it rep- resents the type of event in smart contract log monitored by a node MAn. The address that emitted the event is *address*.

6 | Experiments

6.1 | Experimental Environment

A demo program[2] based on MSC is developed using Go-lang for experimentation. And the smart contract for testing is deployed on the Ethereum Sepolia testnet.[3] Fifteen nodes were used to evaluate the functionality and performance of the program. These 15 nodes are all cloud servers, and their specifications are shown in Table 5.

To validate the feasibility of the framework, we conducted func- tional tests on the program. Additionally, performance testing was carried out to obtain measurable system metrics.

6.2 | Function Test

After deploying the node network, the CLI tool is used to sys- tematically store files of varying sizes, including 1kb, 5kb, 20kb, 50kb, 100kb, 200kb, 500kb, 1000kb, and 2000kb, into the net- work. It is important to note that files larger than 2000kb were not tested, as they exceed the default transfer size limit of the RPC interface.

The demo program was able to create, modify, and revoke data in all the tests, aligning with the expected behavior of MSC.

6.3 | Performance Test

6.3.1 | Time-Consumption

Since nodes in MSC-based applications need to synchronize data after listening to smart contract events, it is necessary to test whether the overall time consumption is within a tolerable range. There are two types of time consumption in MSC for data processing as follows.

- Overall time is the time taken from when the RPC inter- face receives data to the completion of data synchroniza- tion by all nodes. It is the overall time consumption of data processing.
- Synchronize time is the time consumption for all nodes to complete the corresponding operation after monitor- ing *created* or *modified* event. It is a part of the overall time consumption.

Time-consumptions is obtained according to the log informa- tion, the file size and time consumptions line graph is shown in Figure 9. From Figure 9, the overall time has weak correlation with file size, relatively more dependent on the consensus time.

ISSN 2319-5991 www.ijerst.com

Vol. 20, Issue 3, 2024 https://doi.org/10.62643/ijerst.2024.v20.i3.pp353-365

TABLE 5 | Cloud server specifications.

Item	Description
CPU	Intel Xeon Cascade Lake 8255C/Intel Xeon Cooper Lake(2.5GHz/3.1GHz)
Memory	2GB
Bandwidth	10Mbps
System	Ubuntu Server 20.04 LTS x64

The line graph of file size and time consumption of synchro- nize time is shown in Figure 9. The secondary axis of synchro- nization time is on the right side of Figure 9, and its unit is milliseconds. The synchronization time is proportional to the file size. Time consumption is more dependent on the block- chain consensus time as every other blockchain applications. The data storage time consumption of MSC is within the tol- erable range.

6.3.2 | On-Chain Storage Consumption

The gas table encompasses the gas costs associated with various operations in the Ethereum virtual machine (EVM). To accu- rately gauge the storage costs of different on-chain storage meth- ods, the gas table in go-ethereum was adjusted for measurement purposes. In this modification,[4] all non–storage-related costs of operations were standardized to a value of 1. Four types of con- tracts of different data storing methods are deployed on the local test node as follows.

1. Direct Storage: Receive encoded data, then store the data in smart contract.
2. On-chain NFT: Receive a Nonfungible Tokens (NFT) to- ken-id and encoded data, which is subsequently stored into a map within the smart contract. The map is structured as a uint256 to string map, signifying the data resource associ- ated with an NFT.
3. Off-chain NFT: Receive the NFT token-id and IPFS path, which is subsequently stored in the map within the smart contract. The path serves as a reference to the location of the corresponding image in IPFS.
4. MSC: Receive the chameleon hash and random number, then emit event with them as parameters. The chameleon hash corresponds to the index of file in P2P network.

After storing data on these smart contracts respectively, the ex- perimental results are shown in Table 6. Without the impact of computational operation of the gas table, the on-chain storage overhead of storage method can effectively demonstrate by the experimental results. According to the gas usage of direct on- chain storage and on-chain NFT, it is clear that the overhead of direct on-chain storage can be up to 6000 times the size of MSC. In contrast, off-chain storage can significantly reduce storage overhead. The overhead of off-chain storage is indepen- dent of file size due to only index needs to be on-chain storage.

I

ISSN 2319-5991 www.ijerst.com

Vol. 20, Issue 3, 2024 https://doi.org/10.62643/ijerst.2024.v20.i3.pp353-365

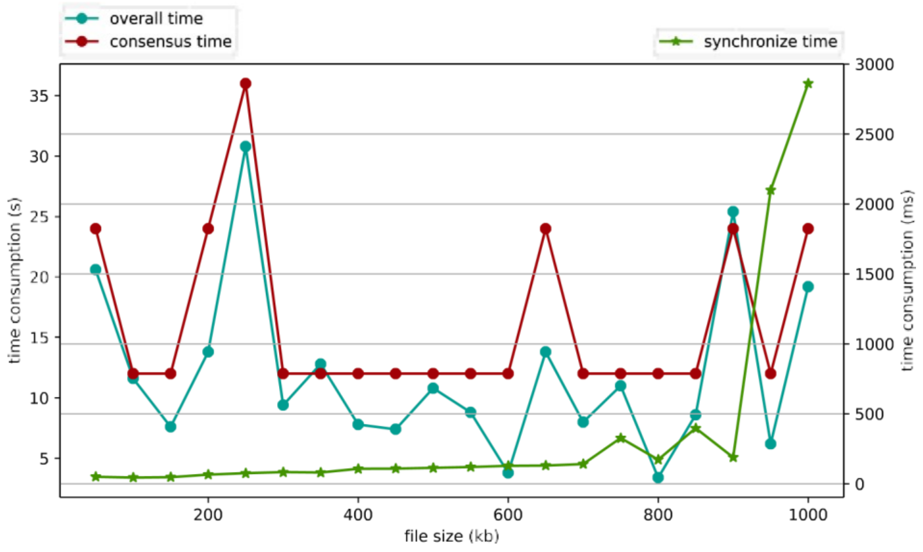


FIGURE 5 | Time consumptions in MSC.

Furthermore, it is apparent that MSC exhibits lower gas con- sumption than Off-chain NFT employing IPFS. Based on the experiments on on-chain storage overhead, MSC has demon- strated its ability to substantially reduce on-chain storage overhead.

6.3.3 | Comparison

The gas usage of MSC and other similar framework while cre- ating, modifying, or revoking data are measured in experiment. SCPKI, IKP, and BGPCoin are used for comparison due to they have similar operations to MSC.

1. Contract creation: All blockchain-based frameworks need to deploy smart contract. Contract creation is the first step in framework initialization.
2. Create data: Insert new data into application. In MSC-based application, data are stored in a small-scale P2P network. In other frameworks, data are stored in IPFS.
3. Modify data: Modify data for a specific index. After modi- fying data in corresponding storage, modify data's onchain index.
4. Revoke data: Remove the corresponding data in applica- tion. Delete data of the corresponding index in storage, and marks the data as being revoked on blockchain.

Comparable processes in SCPKI, IKP, and BGPCoin can be clas- sified into these four operations for the purpose of comparison. The comparison of gas consumption is shown in Figure 10. And to facilitate visualization, the gas usage for contract creation was uniformly reduced (divided by 10). As shown in Figure 10, less gas is consumed in MSC comparing to other frameworks. The gas usage for contract creation related to the implementa- tion of smart contract, it is 119.3% higher than the gas usage in SCPKI. Nevertheless, MSC manages to reduce the on-chain overhead by 31% in comparison. Due to the fact that MSC only

ISSN 2319-5991 www.ijerst.com

Vol. 20, Issue 3, 2024 https://doi.org/10.62643/ijerst.2024.v20.i3.pp353-365

7 | Conclusions

In this research, we presented MSC, a novel data storage system for creating metaverse decentralised applications (dApps) using smart contracts and chameleon hashes. In MSC, the use of chameleon hash allows for the modification of off-chain data without affecting the on-chain index. Transparency in data storage is guaranteed by recording events of metaverse-related smart contracts in blockchain. Furthermore, administrators may alter application code or stored data inside MSC with ease because to the trapdoor function of chameleon hash. Because of the immutability of transactions and the transparency of blockchain, data saved in a P2P network cannot be altered. We have built an MSC-based demo application in Go-lang to test the viability of MSC. The gas table of go-ethereum has been adjusted to assess the actual storage overhead in MSC and other frameworks, allowing us to disregard the influence of other operations. The experimental results show that compared to other frameworks, MSC offers clear benefits, including a significant reduction in on-chain data storage overhead. As a result, MSC may be used to create blockchain-based apps that store data off-chain with much less on-chain overhead.

However, as MSC is dependent on the blockchain's consensus speed, it is unable to meet superior real-time performance on current public chains. So, DNS, PKI, and BGP are the only low-real-time applications that MSC supports. As blockchain technology continues to advance, MSC may be upgraded to operate on blockchain networks designed for high performance or the Internet of Things. The improved MSC, when combined with a high-performance blockchain, will be able to provide very fast services at that time.

References

[1] The paper "Slimchain: Scaling Blockchain Transactions Through Off-Chain Storage and Parallel Processing" was presented at the VLDB Endowment conference in 2021 and can be found in volume 14, issue 11, pages 2314–2326.
The authors are C. Xu, C. Zhang, J. Xu, and J. Pei.
"Smdsb: Efficient Off- Chain Storage Model for Data Sharing in Blockchain Environment" (Springer, 2021), 225–240, is an article by R. Kumar, N. Marchang, and R. Tripathi in the Machine Learning and Information Processing:
Proceedings of ICMLIP 2020 book.
"Certledger: A New PKI Model With Certificate Transparency Based on Blockchain," published in Computers & Security 85 (2019): 333-352, was written by M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar.

[2] In the Proceedings of the ACM Workshop on Blockchain, Crypto-currencies and Contracts (ACM, 2017), M. AlBassam presents "SCPKI: A Smart Contract-Based PKI and Identity System" (pp. 35–40).

[3]"Trustca: Achiev-ing Certificate Transparency Through Smart Contract in Blockchain Platforms" (IEEE, 2020), 1-6. J. Zhao, Z. Lin, X. Huang, Y. Zhang,and S.Xiang. on: HPBDD&IS 2020 International Conference.V. Durham, "Namecoin," Namecoin.org, 2010. 6.

[4] In their article "Assessing Blockchain Consensus and Security Mechanisms Against the 51% Attack," Sayeed and Marco-Gisbert (2019) cite Applied Sciences 9, no. 9: 1788.

[5] ArXiv preprint arXiv:2001.00747, "Improving PKI, BGP, and DNS Using Block- chain: A Systematic Review" (2020), F. S. Ali and A. Kupcu.

[6] In their paper "IKP: Turning a PKI Around With Decentralised Automated Incentives," published in 2017 at the
IEEE Symposium on Security and Privacy (SP) (IEEE, 2017), S. Matsumoto and R. M. Reischuk discuss this topic in length.

[7] [In the presentation "A Blockchain-Based PKI Management Framework" given at the First IEEE/IFIP International Workshop on Managing and Managed by Blockchain (Man2Block) Colocated with IEEE/IFIP Noms 2018, which took place in Tapei, Taiwan from April 23-27, 2018, pages 1-6, authors A. Yakubov, W. Shbair, A. Wallbom, and D. Sanda were recognised.

ISSN 2319-5991 www.ijerst.com

Vol. 20, Issue 3, 2024 https://doi.org/10.62643/ijerst.2024.v20.i3.pp353-365

[8] In the paper "A Blockchain-Based Decentralised Data Storage and Access Framework for Pinger" published in the 2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (Trustcom/Bigdatase) (IEEE, 2018), pages 1303– 1308, the authors provide a blockchain-based solution for decentralised data storage and access.

[9] "A Secure Cloud Storage Framework With Access Control Based on Blockchain," published in 2019 by IEEE Access, was authored by S. Wang, X. Wang, and Y. Zhang.

[10]In "A Data Storage Method Based on Blockchain for Decentralisation DNS," J. Liu, B. Li, L. Chen, M. Hou, F. Xiang, and P. Wang present their work (in