DSChain: A Blockchain System for Complete Lifecycle Security of Data in Internet of Things

Jie Cui, Yatao Li, Qingyang Zhang, Hong Zhong, Chengjie Gu, and Debiao He

Abstract—There is a growing concern about the complete lifecycle security of data in Internet of Things (IoT). This may cause privacy and trust problems for users regarding data sources, data storage, and access control for data sharing. Blockchain is a valuable solution to the above problems through distributed ledger technology, and it has been widely applied in various fields such as public services, finance, and IoT. However, the data in IoT are characterized by a large quantity, large capacity, and timely response, and existing blockchain systems only partially resolve them for data security and performance. We propose DSChain for IoT data security to address the challenges mentioned above. Our system uses a certificateless signature to ensure a trusted data source and public auditing to ensure the integrity of stored data while using ciphertext-policy attribute-based encryption to control access to shared data. Moreover, we propose a packaging mechanism based on the Merkle Hash Tree that effectively improves system performance. We implement the DSChain and provide a detailed analysis of performance and security. The experimental results indicate that DSChain can achieve approximately 1,035 transactions per second on a single peer and is scalable.

Index Terms—Blockchain, IoT, certificateless signature, ciphertext policy, ABE, public auditing, data lifecycle security.

1 Introduction

MANY smart terminals have emerged with the rapid growth of the Internet of Things (IoT). Gartner predicts there will be 40 billion terminals in the IoT by 2025, and the amount of data collected by terminals will exceed 80 ZB (1 ZB = 10^{21} bytes)[1]. The conventional approach uses centralized services (e.g., private cloud) to manage terminals and their data, which may increase the risk of misuse and tampering with sensitive data. In addition, uncontrolled access to these data leads to critical data security risks for end users [1], which is unfavorable to the development of IoT. Therefore, it is of great concern for the complete lifecycle security of data.

Recently, the decentralized blockchain technology on which Bitcoin [2] relies has attracted significant interest from industries and academia. A blockchain is a decentralized distributed database that links data blocks in an orderly pattern and guarantees that the data are tamper-proof and unforgeable through cryptographic techniques. Blockchain technology has evolved to the 3.0 version [3], which brings prospects for data security management in IoT [4]. Therefore, integrating blockchain and IoT [5]–[7] is gradually becoming accepted.

- *J. Cui, Y. Li, Q. Zhang and H. Zhong are with the Key Laboratory of Intelligent Computing and Signal Processing of Ministry of Education, School of Computer Science and Technology, Anhui University, Hefei 230039, China, and the Anhui Engineering Laboratory of IoT Security Technologies, Anhui University, Hefei 230039, China (E-mail: zhongh@ahu.edu.cn).*
- *C. Gu is with the School of Public Security and Emergency Management, Anhui University of Science and Technology, Hefei, 231131, China and the Security Research Institute, New H3C Group, Hefei, 230088, China (E-mail: gcj@ustc.edu.cn).*
- *D. He is with the School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China and the Shanghai Key Laboratory of Privacy Preserving Computation, MatrixElements Technologies, Shanghai 201204, China (E-mail: hedebiao@163.com).*

1. https://www.gartner.com/en/documents/4000845

Various schemes have attempted to apply blockchain technology to the IoT field [8]–[10]. There are two well-known types of schemes for building IoT applications using blockchain. One is to build IoT applications based on existing blockchain systems, such as Ethereum [11] and Hyperledger Fabric [12], which provide extended code instruction execution environments. The code instruction, smart contract in Ethereum and chaincode in Fabric, is deployed in the blockchain system and executed by a virtual machine, which allows users to program business logic [13]. However, they are limited by existing blockchain systems; for example, Ethereum limits the block size to a few megabytes (MB) and gas spent, and Fabric can only store data on-chain. The other is a native blockchain system built for IoT, such as WaltonChain [14], IoTeX [15], and IOTA [16], which are to address specific challenges. However, they weaken the security features of blockchain and are not universally applicable. Therefore, although the integration of blockchain and IoT is rapidly developing and being accepted, some challenges still cannot be neglected [17].

Trust and Privacy: The openness, transparency, and immutability of blockchain can be a valuable solution for identity privacy management and building trust [18]. Many IoT applications require users to provide sensitive data (e.g., smart grid) for additional services, which puts the data at risk of misuse. However, while users trust each other, they also want their sensitive data to be protected [19].

Storage Capacity and Data Integrity: Some blockchain systems limit block size to a few MBs to reduce the latency of accessing transactions. However, terminals usually generate gigabytes (GB) of data at an actual time, and the capacity limitation of block poses a considerable obstacle to the integration of blockchain and IoT. Some solutions utilize centralized servers to store large amounts of data. However, not all centralized servers are trusted and may tamper with data for their interests [20].

Performance and Availability: The traditional chain-structured blockchain uses a decentralized consensus mechanism, resulting in low throughput and unsuitable for large-scale IoT applications. However, the novel directed acyclic graph (DAG) structure blockchain improves process rate by recording transactions asynchronously and concurrently, which makes it unable to guarantee the ultimate consistency of data simultaneously [21].

To address the above challenges, we design DSChain, a blockchain system, to ensure the complete lifecycle security of IoT data by integrating the certificateless signature (CLS), ciphertext-policy attribute-based encryption (CP-ABE), and public auditing (PA) paradigms. Data is stored using a combination of on-chain and off-chain, improving the storage capacity of our system. In addition, we use the Merkle Hash Tree (MHT) structure for packaging multiple messages into a transaction to improve system performance.

Our work makes the following contributions:

1) We clarify three significant challenges in the integration of blockchain and IoT. Accordingly, we propose DSChain, a blockchain system, and its integration scheme that integrates CLS, CP-ABE, and PA paradigms to ensure the complete lifecycle security of IoT data, which includes (i) a trusted source of collected data; (ii) access control for shared data; (iii) integrity of stored data. And the instances of paradigms are substitutable.
2) We propose an MHT-based packaging mechanism that uses message queue (MQ) middleware to preprocess service requests and trigger packaging according to a packaging policy. It satisfies the storage requirement for large amounts of real-time data and realizes lower latency and higher throughput. Meanwhile, combining on-chain and off-chain storage improves storage capacity and user privacy.
3) We implement the DSChain and provide a detailed analysis of its performance and security. The experimental results indicate that DSChain can achieve approximately 1,035 transactions per second (tps) on a single peer and less than 200 ms data processing latency. In addition, our system is scalable and can be deployed using clusters to accommodate large-scale IoT applications.

The remainder of this paper is organized as follows. We analyze the motivation in Section 2 and then provide preliminaries in Section 3. We describe the design of DSChain in Section 4 and our integration scheme in Section 5. In Section 6, we introduce our system mechanism. We provide a full implementation of the DSChain in Section 7. We present the performance evaluation and security analysis in Section 8 and review related work on blockchain systems in Section 9, followed by concluding remarks in Section 10.

2 Motivation

IoT and blockchain technologies have recently rapidly developed and integrated with growing market demand. As the data are the oil of IoT development, ensuring its complete lifecycle security is crucial. In this section, we analyze data security issues in IoT and review some blockchain systems, then analyze the challenges and our design goals.

2.1 Data Security Issues

The IoT is a network that connects physical terminals to a network so that they can upload and exchange data. Data security management during the collection, storage, and sharing phases is a widely publicized issue. However, the inherent features of blockchain are underpowered in their suitability to IoT applications [22].

2.1.1 Data Collection Issue

The data collected by terminals can make services more efficient and benefit enterprises. Terminal manufacturers usually use centralized services to manage mass terminals and data, increasing the difficulty and cost of maintenance. However, users need to trust the manufacturers completely [23]. Some manufacturers utilize a decentralized blockchain system and certificates to guarantee a trusted data source. However, managing mass certificates increases maintenance difficulty and reduces performance [24]. The CLS without public key certificates could simplify certificate management and facilitate deployment on resource-constrained IoT devices. In addition, the key generation center (KGC) only stores the partial private key of users to enable traceability, which can guarantee a trusted data source. The public-private key pairs of users are generated locally and are not accessible by KGC, protecting user privacy.

2.1.2 Data Storage Issue

With the massive emergence of data, it needs to be stored securely and efficiently. Otherwise, the data value will be vastly reduced. The conventional approach uses a centralized cloud storage service to manage the data. However, its providers may tamper with data due to interests or mischief, and the integrity and truthfulness of the data cannot be guaranteed. The tamper-proof feature of blockchain is suitable for solving centralization and trust problems [25]. However, the mass data stored on-chain increases its storage pressure. Some blockchain systems use a combination of on-chain and off-chain data storage to solve data capacity problems [26]. To prevent off-chain sensitive data from being viewed by unauthorized entities, it is necessary for the blockchain system to provide the confidentiality of data. In addition, the tamper-proof feature of blockchain is underpowered to protect off-chain data, which could be at risk of tampering. Users prefer to have the ability to check the integrity of their data proactively or to regulate the behavior of storage service providers through PA. A blockchain-based PA for off-chain data could solve the trust problem for users [27].

2.1.3 Data Sharing Issue

Data exchange conveys the data value [28], meaning that data sharing effectively eliminates information isolation. However, users expect data to be reasonably viewed and used by permitted entities. When sharing data using a traditional public key cryptosystem, the user must perform multiple encryption operations for all receivers. Attribute-based encryption (ABE) can avoid multiple encryption operations on the same data, instead using attributes to specify the access policy for the ciphertext. Generally, access control

TABLE 1
The Comparisons of Different Blockchain Systems.

System	Type	Features	Throughput	Data Lifecycle Security				
				Trusted Source	Confidentiality	Access Control	Off-chain Storage	Integrity
HIBEChain [31]	Ethereum-based	Gas Limit	32,000 tps[2]	✔	✘	✘	✘	✘
FabZK [32]	Fabric-based	Permissibility	Less Than Fabric	✘	✔	✘	✘	✔
PrivySharing [33]	Fabric-based	Permissibility	Less Than 200 tps	✘	✔	✔	✘	✘
WaltonChain [14]	Native	Multi-chain	100 tps	✔	✔	✔	✔	✘
IoTeX [15]	Native	TEE	–	✔	✘	✘	✘	✘
IOTA [16]	Native	Tangle Network	600 tps	✘	✘	✘	✘	✘
Our Goals	Native	Cryptographic Paradigms	Higher	✔	✔	✔	✔	✔

[1] The tps denotes transactions processed per second.
[2] The tps reaches 32,000 for the 6-ary case with 259 blockchains and 2,590 validators and reaches 170 for a leaf blockchain with (3,4)-threshold.

management relies on a central trusted entity, and data security is at risk. The decentralization and traceability of blockchain can solve trust problems of data sharing, which is beneficial in resisting a single point of failure [29]. In contrast, users prefer fine-grained access control over their sensitive data [30]. CP-ABE allows the user to specify an access structure whose attributes have logical relationships that can be used for fine-grained and more flexible access control. In addition, CP-ABE can effectively improve the confidentiality and security of shared data.

2.2 Existing Schemes

We analyze the security issues of IoT data at different phases in Section 2.1. We know that a trusted data source is a prerequisite for data security; expanding on-chain storage can effectively increase the blockchain storage capacity; the confidentiality and integrity of the stored data can guarantee the privacy of users; and users expect more permission control over shared data. Therefore, we compare different blockchain systems proposed by others to solve the above problems. The result is illustrated in TABLE 1.

2.2.1 Based on Existing Blockchain Systems

Some blockchain systems provide extended code instruction execution environments, which allow users to build IoT applications to manage data securely. Wan et al. [31] proposed an Ethereum-based HIBEChain to solve the terminal identity problem in large-scale IoT. However, it is limited by the block size and gas limitation of Ethereum, which leads to long transaction confirmation latency and does not support unstructured data storage that occupies large storage space. Kang et al. [32] and Makhdoom et al. [33] proposed Fabric-based blockchain systems, where the former achieves encrypted data storage and auditability, and the latter achieves data access management by embedding access control rules in smart contracts. However, they are limited by Fabric, which puts heavy storage pressure to store huge amounts of data on-chain.

2.2.2 Native Blockchain Systems for IoT

To address the specific challenges in IoT, some organizations and institutions autonomously design and implement native blockchains for IoT. Waltonchain team [14] proposed to use cross-chain technology to build an IoT business ecosystem for trusted data sources, private chain data sharing, and data traceability. However, it has weak performance and does not support data integrity verification. IoTeX team [15] proposed to ensure that the data source is trusted and enhances user privacy by decentralizing identity. However, it does not guarantee data confidentiality or support off-chain storage. To improve blockchain system throughput, Silvano et al. [16] proposed to use a Tangle network with the DAG structure to confirm transactions. However, it reduces the system's security and does not support off-chain storage.

2.3 Challenges and Goals

The above issues are usually addressed by integrating multiple cryptographic protocols, such as CLS, CP-ABE, and PA, which will increase the complexity of system design under different regimes. It is crucial to design a scheme that can integrate the above cryptographic protocols to accommodate complex application scenarios. In addition, the blockchain system performance is also worth our attention. The integration scheme may face challenges such as reuse parameters of different cryptographic protocols, authentication of requests, and workflow optimization. It will also be a considerable challenge to support the pluggability and scalability of cryptographic protocols at the system level. Meanwhile, clarify system participants and their functions to streamline workflows.

With the above challenges in mind, we summarize the design goals to provide a system for the complete lifecycle security of IoT data (see TABLE 1). Use decentralized blockchain technology and pluggable cryptographic paradigms to ensure a trusted data source, the integrity and confidentiality of stored data, and fine-grained access control for shared data. Meanwhile, it provides a solution for massive data storage while ensuring system performance for large-scale IoT applications.

3 Preliminaries

This section provides the basics on which our system relies, including the Merkle Hash Tree and some systematic terminology concepts.

3.1 Merkle Hash Tree

The Merkle Hash Tree (MHT) is a hash binary tree with at most two subtrees per node, where the subtree is usually named left subtree or right subtree. A leaf node of an MHT is the hash of a data block, and a non-leaf node is the hash of sub-nodes, where each node represents a piece of structured data. The MHT has excellent features such as integrity verification and fast data querying. Its structure is described below:

- The MHT is a binary tree structure with all the characteristics of a tree structure.
- The user inputs the value of the leaf node in MHT. For example, the hash of the data block is used as the value of the leaf node.
- The value of a non-leaf node is computed from the value of its sub-nodes, usually using a specified hash algorithm (e.g., SHA256 algorithm).

3.2 Terminology Concepts

We provide some systematic terminology concepts below to make it more convenient to understand the design and workflow of our system.

Channel: A channel is a specialized virtual subnet for communication among peers and is used for private and confidential transactions. The transactions in the blockchain network (BCN) will only be transmitted and executed within a single channel. It uses logical isolation techniques to ensure that data, transactions, and nodes among channels are invisible. The manager creates a channel, and only trusted peers can join.

Message Queue: The message queue (MQ) is an inter-application communication mode where messages can be returned immediately after they are sent, and the messaging system ensures reliable delivery of the messages. The queue is a data structure that satisfies first-in-first-out, which makes these messages strictly sequential. Its message retry mechanism ensures the reliability of messages, while the atomicity of delivering messages and the acknowledgement mechanism ensure the ultimate consistency of distributed transactions.

Publish-Subscribe Service: The publish-subscribe service is a messaging paradigm based on MQ. Instead of sending messages directly to specific recipients, the sender divides them into different categories for recipients (i.e., subscribers) to subscribe to. Similarly, subscribers can subscribe to one or more categories of messages without caring about the identity of the publisher.

Topic: The topic-based subscription is one of the ways to receive categorized messages. All messages in the topic will be pushed to its subscribers. Subscribers who subscribe to the same topic will receive the same messages.

4 DSChain Design

This section presents an overview of the proposed system, the transaction execution flow, the threat model, the hierarchical structure, and the consensus service.

4.1 Overview

We design the prototype blockchain system to guarantee the complete lifecycle security of IoT data. Fig. 1 presents our system model. It consists of three entities, which are described in detail below.

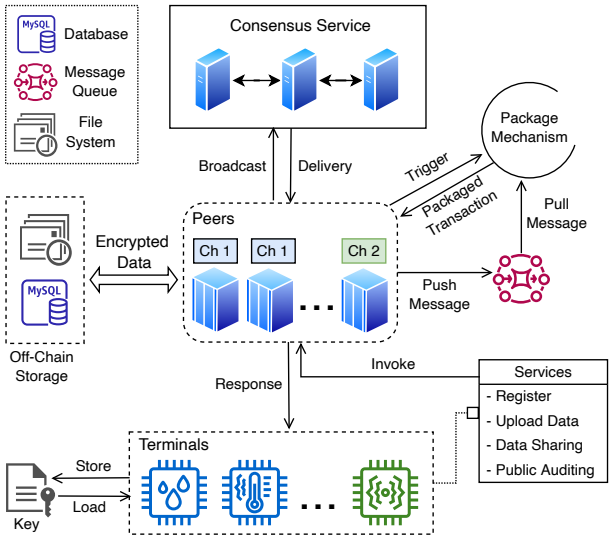


Fig. 1. System model.

- *Terminal:* Terminals are smart IoT devices with finite computing and communication capabilities. Each terminal has a unique decentralized identity (DID) in the whole network and requests identity registration from peers to obtain its partial key. Only authorized terminals can connect to the BCN and send service requests to peers.
- *Peer:* Peers run on cloud-based physical machines with high-resource capacity, high performance, and strong computing power and undertake most of the computation of the BCN. Peers maintain key services, including terminal identity registration, data upload, data sharing, and public auditing. Usually, the peers in a channel jointly process requests from terminals, broadcast packaged transactions to consensusers, and eventually append successful transactions to the local ledger.
- *Consensuser:* The consensusers collectively maintain the consensus service and a system chain. The consensus service supports the logical isolation for channels and ensures the ultimate consistency of transactions on a given channel. The system chain stores configuration information for the entire BCN, and system managers can dynamically update the configuration.

4.2 Transaction Execution Flow

All peers and consensusers maintain the BCN to provide services for terminals. For better understanding, we define a complete transaction execution flow from the terminal invoking the service to the BCN confirming the transaction. We divide it into three parts, as shown in Fig. 2.

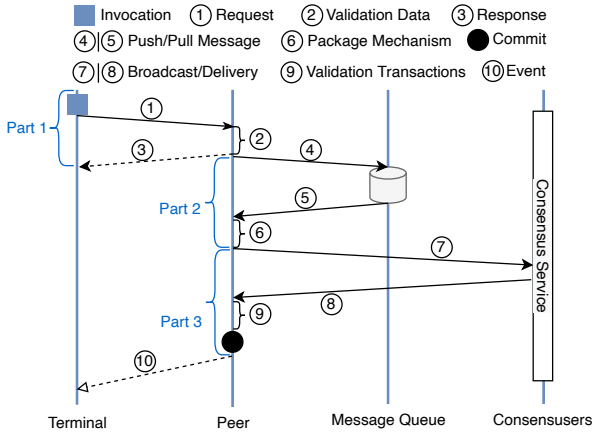


Fig. 2. Transaction execution flow.

Part1: Before joining the BCN, the terminal submits DID to the peer for identity registration ① and obtains a key for its unique identity. For secure data sharing, the terminal encrypts the raw data before uploading and embeds access control rules in the ciphertext so that only entities that satisfy the rules can decrypt the encrypted data. The terminal uses the key to sign the encrypted data to enable data traceability. Terminals can trigger public auditing ①, which calls for peers to verify the integrity of stored data and submit the result to the BCN. Once peers receive service requests, they verify the validity of requests based on the registration information of terminals ② and return the result ③.

Part2: Peers are responsible for processing service requests and submitting the packaged transaction to the consensus service. The peer maintains a database and a file system to store the encrypted data, providing off-chain storage services. When the peer receives a request from the terminal to upload data, the encrypted data is stored off-chain, and its key information msg is pushed into the MQ ④. Once the condition of packaging is satisfied, the peer pulls a fixed amount i of messages $MSG = \{msg_1, msg_2, \ldots, msg_i\}$ from the MQ ⑤. Then, the peer packages MSG, channel information, and the peer's signature into a single transaction tx using the MHT-based packaging mechanism ⑥.

Part3: The peer broadcasts the tx to the consensus service ⑦. Peers usually subscribe to the topic $topic$ of the latest block and transaction information on their channel. When the consensus service reaches a consensus on transactions, they package all transactions into a block. The consensusers delivery blocks to peers by a publish-subscribe service ⑧. Peers verify the validity of transactions after receiving the latest block and append valid transactions to the local ledger ⑨. Meanwhile, peers also provide a publish-subscribe service for transactions to terminals ⑩.

4.3 Threat Model

In the BCN, the consensusers are not associated with specific user data and only package transactions into blocks through the consensus mechanism. Therefore, we assume that the consensusers are fully trusted and cannot be compromised. The adversaries considered in our system can be classified into internal and external adversaries. Internal adversaries are entities directly involved in the BCN, i.e., peers and terminals, and external adversaries are entities not directly involved in the BCN. Both internal and external adversaries can initiate passive and active attacks. When initiating a passive attack, an attacker may continuously obtain some encrypted messages in the BCN and attempt to decipher them. In an active attack, an attacker can access the data in the channel, read, intercept, replay, modify, and forge the data. In addition, an attacker can forge the identity of terminals to initiate a DDoS and Man-in-the-Middle attack.

4.4 System Hierarchical Architecture

We design a system hierarchical architecture divided into five layers, as shown in Fig. 3. As a whole, they provide underlying services for the BCN.

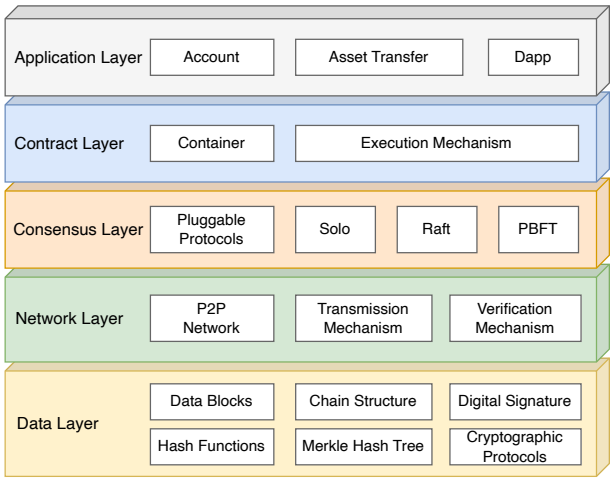


Fig. 3. System hierarchical architecture.

- *Data Layer:* The essence of blockchain is a decentralized database, and the data layer is the infrastructure for reliable storage. It links data blocks with a chain structure in an orderly pattern and ensures data security and tamper-evidence through digital signatures and cryptographic protocols. Meanwhile, hash functions and MHT provide the ability to verify and query transactions quickly.
- *Network Layer:* All nodes in the BCN compose a peer-to-peer (P2P) network. Transactions are transferred among nodes using the transmission mechanism, which ensures reliable data transmission. The validation mechanism is used to verify the validity of data to ensure system security.
- *Consensus Layer:* The consensus layer provides the pluggable capability of the consensus mechanism. It provides several consensus mechanisms for managers, such as Solo, Raft [34], and PBFT [35]. The consensus mechanism uses MQ to ensure transaction consistency in the entire BCN, which improves the success rate of transaction. This layer only sequences transactions and packages them into a block.
- *Contract Layer:* The contract layer extends the blockchain system, making the blockchain programmable. Smart contracts consist of automated

script code instructions that execute various business logic within the constraints of the system. They run in containers mutually isolated from peers to ensure trusted execution environments.

- *Application Layer:* Through this layer, the system can be associated with various industries (e.g., IoT) and use decentralized computing power to build trust among entities. Individuals or organizations can customize execution logic to extend the system according to diverse business requirements, such as account management and asset transfer.

4.5 Consensus Service

All consensusers collectively establish a general sequence of transactions through the consensus mechanism. The consensus service receives the arbitrary transaction tx broadcast by the peer and verifies its validity. The valid tx is immediately published to the MQ maintained by the consensus service. The consensus policy is specified by the system manager when initializing the BCN and configured in the system chain. It can be set as follows: the total size of all transactions (e.g., 6MB), the number of transactions (e.g., 100), or the packaging timeout time (e.g., 2s). The manager can also dynamically adjust the consensus policy. Once the consensus conditions are satisfied, the assigned consensuser consumes a list of transactions $TX = \{tx_1, tx_2, \ldots, tx_j\}$ (j denotes the number of transactions) from the MQ. With no concern for the execution logic of TX, the consensuser just packages the TX into a $block$ with a specified height. The $block$ also contains the consensuser's signature and channel information. The consensus service provides a publish-subscribe service, and peers can subscribe to the $topic$ of their channels. Then, it delivers the $block$ to all peers in the corresponding channel. Notably, the consensus service is usually deployed using clustering to ensure its liveness and scalability.

5 Construction of Integration Scheme

In this section, we introduce the CLS, CP-ABE, and PA paradigms, which are generic and not limited to a specific scheme. Then, we construct a data security integration scheme using specific instances of the above paradigms as samples.

5.1 Certificateless Signature

The certificateless signature (CLS) is a novel type of public key cryptosystem. It not only retains the advantage of the identity-based public key cryptosystem (ID-PKC), which does not require public key certificates; but also solves the key management problems inherent to ID-PKC and is easy to deploy on resource-constrained devices.

A CLS paradigm involves three entities: the key generation center (KGC), the signer, and the verifier. There are seven algorithms in a CLS paradigm, as described below.

1) *CLS.Setup:* The KGC initializes the system by running this algorithm. On inputting a security parameter λ, the algorithm outputs a system master key msk and the system public parameters PP.
2) *CLS.ExtPartialPvtKey:* On inputting the master key msk, the public parameters PP, and ID of the user, the KGC computes the partial private key d and sends it to the user through a secure channel.
3) *CLS.SetSecretValue:* The user takes as input the public parameters PP and outputs a secret value x.
4) *CLS.SetPubKey:* Users execute the algorithm. Taking as input the public parameters PP, the secret value x, and outputs the public key PK.
5) *CLS.SetPvtKey:* Taking as input the public parameters PP, the partial private key d, and the secret value x, the algorithm returns the private key SK.
6) *CLS.Sign:* As input a message m, the public parameters PP, the signer's ID, and the private key SK, the signer calls this algorithm to generate a signature σ.
7) *CLS.Verify:* The verifier runs this algorithm by receiving the signature σ, the public parameters PP, the signer's ID, the signer's public key PK, and the message m. It outputs 1 or 0 to indicate whether the signature is valid.

5.2 Ciphertext-Policy Attribute-Based Encryption

Attribute-based encryption (ABE) associates a ciphertext or key with an attribute set and access structure, which can be decrypted only if the attribute set satisfies the access structure. According to this pairwise correspondence, ABE can be divided into key-policy attribute-based encryption (KP-ABE) and ciphertext-policy attribute-based encryption (CP-ABE). However, CP-ABE is well adapted to build secure data-sharing solutions in cloud environments.

There are four algorithms in a CP-ABE paradigm, as described below.

1) *CPABE.Setup:* The algorithm takes as input a security parameter λ. It outputs the public parameters PK and a master key MK.
2) *CPABE.Encrypt:* Taking as input the public parameters PK, the message M, and an access structure $\mathbb{A}$ over the universe of attributes, the algorithm encrypts M and produces a ciphertext CT. Assume that the ciphertext implicitly contains $\mathbb{A}$.
3) *CPABE.KeyGen:* The key generation algorithm inputs the master key MK and a set of attributes $\mathbb{S}$ that describe the key. It outputs a private key SK.
4) *CPABE.Decrypt:* The algorithm inputs the public parameters PK, a ciphertext CT containing an access structure $\mathbb{A}$, and a private key SK for a set of attributes $\mathbb{S}$. If the set of attributes $\mathbb{S}$ satisfies the access structure $\mathbb{A}$, then the algorithm will decrypt the ciphertext and return a message M.

5.3 Public Auditing

Public auditing (PA) is the verification of data integrity as a way to ensure the security of stored data. Blockchain makes it possible to automate audits and improve the efficiency and quality of audits.

A blockchain-based PA scheme involves the data owner (DO) and the cloud service provider (CSP). There are six algorithms in a PA paradigm, as described below.

1) *PA.Setup:* The DO takes as input a security parameter λ and outputs a public-private key pair (PK, SK).
2) *PA.TagGen:* The DO runs the algorithm, using the private key SK, the identity U_{ID}, and the encrypted file F' as input, and outputs the tags τ of the F'.
3) *PA.Store:* The DO signs the encrypted file F' as Φ using SK and uploads them to the CSP.
4) *PA.ChalGen:* Run it to input the public key PK and the identity U_{ID} to generate the challenge $chal$.
5) *PA.ProofGen:* The algorithm can be executed by a public auditor (another DO). The public auditor requests the CSP to construct an MHT based on the challenge value $chal$ and the file B_{ID} and generates the proof R_{csp}. Then, the public auditor queries the records on the blockchain, constructs an MHT by the tags of the encrypted files, and generates the proof R_{bc}. Finally, the public auditor uses PK to sign $\{R_{csp}, R_{bc}\}$ and returns it to the DO.
6) *PA.VerifyProof:* The DO checks whether $R_{csp} = R_{bc}$. If the equation holds, the data is integrity; otherwise, the data is corrupted.

5.4 Our Integration Scheme

The above three cryptographic paradigms can address the following issues: the trusted source of collected data, the fine-grained access control of shared data, and the integrity of stored data. To ensure complete lifecycle security of IoT data, our scheme integrates the CLS, CP-ABE, and PA paradigms mentioned above. Remarkably, our system supports arbitrary specific schemes (instances) that satisfy the above paradigms. In other words, we focus on the functionality of instances rather than their design details. System managers may replace the corresponding instances during the compilation and deployment phases.

With that foundation, we clarify the system participants and their functions and improve the interaction efficiency by reusing public parameters and simplifying the system workflows. We construct our data security scheme by integrating Jia et al.'s CLS scheme [36], Waters et al.'s CP-ABE scheme [37], and Li et al.'s PA scheme [38] as an example. The following is a description of the specific example.

1) *DS.SystemInit:* Given a security parameter λ, choose a bilinear group $\mathbb{BG} = (\mathbb{G}, \mathbb{G}_T, p, g)$, where $\mathbb{G}$ and $\mathbb{G}_T$ are two multiplicative cyclic groups of prime order p, and g is a generator. Choose two distinct secure hash functions: $h : \{0,1\}^* \rightarrow Z_p^*$ and H is the SHA256 hash algorithm. The peer chooses a random number $\alpha \in Z_p^*$ and calculates the system public key $spk = g^\alpha$. Return the system public parameters $PP = \{\mathbb{BG}, spk, h, H\}$ and the system secret key $ssk = \alpha$.
2) *DS.Register:* Taking as input the system master key ssk, the public parameters PP, and the terminal DID did_i, the peer runs the $CLS.ExtPartialPvtKey(ssk, PP, did_i)$ algorithm to get the partial private key d_i and returns it.
3) *DS.GenKeyPair:* Taking as input the public parameters PP, the terminal DID did_i, and the partial private key d_i, the terminal runs the $CLS.SetSecretValue(PP, did_i)$ algorithm to get the secret value x_i, and then runs the $CLS.SetPvtKey(PP, d_i, x_i)$ algorithm and $CLS.SetPubKey(PP, x_i)$ algorithm to obtain the public-private key pair (sk_i, pk_i).
4) *DS.Sign:* Taking as input the message M, the public parameters PP, the terminal DID did_i, and the private key sk_i, the terminal runs $CLS.Sign(M, PP, did_i, sk_i)$ to get a signature σ.
5) *DS.Verify:* Taking as input the signature of message σ, the public parameters PP, the terminal DID did_i, the public key pk_i, and the message M, the peer runs the $CLS.Verify(\sigma, PP, did_i, pk_i, M)$ algorithm to verify the validity of the signature and return the result $\{Result\}$.
6) *DS.Encrypt:* Taking as input the public parameters PP, the message M, and the access structure $\mathbb{A}$, the terminal runs the $CPABE.Encrypt(PP, M, \mathbb{A})$ algorithm to get the ciphertext CT of M, where CT contains $\mathbb{A}$. In running the $CPABE.Encrypt$ algorithm, the key is generated based on a random function and uses a one-time pad (OTP), where $key \in Z_p^*$. Encrypt the data using the AES symmetric encryption algorithm and return CT.
7) *DS.GenDecryptKey:* Taking as input the system master key ssk and the set of attributes $\mathbb{S}$ of the data receiver, the peer runs the $CPABE.KeyGen(ssk, \mathbb{S})$ algorithm to compute the key key and returns it.
8) *DS.Decrypt:* Taking as input the public parameters PP, the ciphertext CT, and the key key, the data receiver runs the $CPABE.Decrypt(PP, CT, key)$ algorithm to decrypt the ciphertext CT. If the set of attributes $\mathbb{S}$ corresponding to key satisfies $\mathbb{A}$, the message M can be successfully decrypted.
9) *DS.GenTag:* This algorithm generates audit tags for the encrypted unstructured data CT. We know that $CT = \{C_1, C_2, \ldots, C_n\}$. The terminal runs the $H(C_i)$ hashing algorithm to get the hashtag τ_i, where $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Return the hashtag τ.
10) *DS.GenChal:* Taking as input the public key pk_i and the terminal DID did_i, the terminal runs the $PA.ChalGen(pk_i, did_i)$ algorithm to generate the challenge $chal$ and returns it.
11) *DS.ProofGen:* The audited node queries the locally stored file based on the challenge value $chal$ and the file cid_{root}, constructs the MHT for the cid_{root} and generates the proof R_{peer}. The public auditor (that is different from the audited node) queries the records on the blockchain based on the $chal$ and cid_{root}, constructs the MHT by the hashtag of encrypted files, and generates the proof R_{bc}. The result R_{peer} and R_{bc} are signed using pk_i of the terminal did_i and returned as a result. The R_{peer} and R_{bc} are signed by the audited node and the public auditor, respectively, and returned as results.
12) *DS.VerifyProof:* The user (i.e., terminal) checks whether $R_{peer} = R_{bc}$ and returns the result $Result$.

6 System Mechanism

In this section, we describe the overall work of the system mechanism in detail, as shown in Fig. 4. It contains six key phases and covers the complete lifecycle of data.

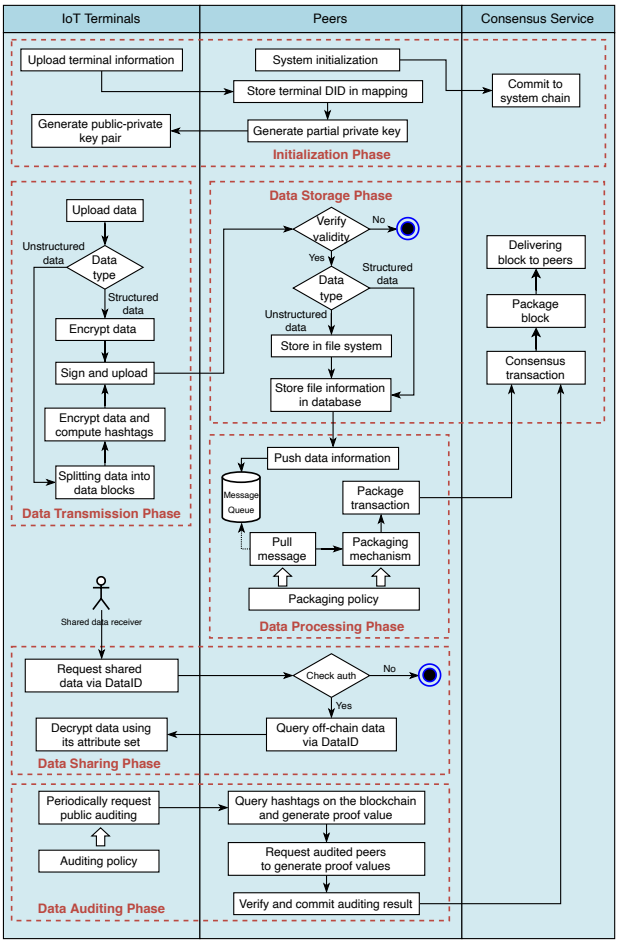


Fig. 4. The overall work of system mechanism.

6.1 Initialization Phase

Assumes a secure channel is established between terminals and peers to exchange data and keys.

6.1.1 System Initialization

The authorized peer can create a channel, then run the $DS.SystemInit(\lambda)$ algorithm to obtain the public parameters $PP = \{\mathbb{G}, \mathbb{G}_T, p, g, spk, h, H\}$ and the master key ssk. The peer will publish PP to the system chain and keep ssk secret. Other authorized peers can join the created channel and run the $DS.SystemInit(\lambda)$ algorithm to do the same thing. Remarkably, all peers within the channel can synchronize the data from the system chain through a transmission mechanism.

6.1.2 Terminal Registration

The terminal has a DID did_i and identity-related material. $\cup_{i=0}^{\infty} did_i \subseteq \mathbb{DID}$, where $\mathbb{DID}$ is a superset of all terminals that are connected to the BCN. Before connecting to the BCN, the terminal sends an identity registration request to the peer. The peer runs the $DS.Register(ssk, PP, did_i)$ algorithm to generate a partial private key d_i and return it to the terminal. Peers maintain a global mapping between the partial private key and the terminal identity $ide : did_i \rightarrow d_i$, which is shared with other peers in the channel through the transmission mechanism. Then, the terminal runs the $DS.GenKeyPair(PP, did_i, d_i)$ algorithm to compute the public-private key pair (sk_i, pk_i) and manages it by placing it in a secure key container or updating it periodically to prevent a compromised-key attack.

6.2 Data Transmission Phase

To protect confidentiality and secure access control for shared data, we encrypt data using the CP-ABE. According to data sharing needs, terminals define the access structure $\mathbb{A}$. The data can be decrypted if the entity's attributes satisfy $\mathbb{A}$. In addition, we use the CLS to ensure a trusted source of data and its integrity. The raw data generated from the terminals is usually in the form of stream data, which will be processed by the edge gateway or edge computing device into structured data (e.g., body temperature) and unstructured data (e.g., surveillance videos).

Structured Data: In our system, structured data $data_S$ represents data that occupies little storage capacity and is easy to express. For secure data sharing, the terminal runs the $DS.Encrypt(PP, data_S, \mathbb{A})$ algorithm to obtain the ciphertext CT. Then the terminal runs the $DS.Sign(CT, PP, did_i, sk_i)$ algorithm to obtain the signature σ_t and uploads $\{CT, \sigma_t\}$ to the peer.

Unstructured Data: Unstructured data $data_U$ represents data that occupies a large storage capacity and is difficult to express. Therefore, the terminal splits $data_U$ into n data blocks $F = \{f_1, f_2, \ldots, f_n\}$ to facilitate transfer and storage. The terminal runs the $DS.Encrypt(PP, F, \mathbb{A})$ algorithm to encrypt the F separately to get the ciphertext CT, where $CT = \{C_1, C_2, \ldots, C_n\}$. Then the terminal runs the $DS.Sign(CT, PP, did_i, sk_i)$ algorithm to sign CT separately and obtains the signature σ_t, where $\sigma_t = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$. For auditing purposes, the terminal runs the $DS.GenTag(CT)$ algorithm to obtain the hashtag τ, where $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. After data processing, the terminal uploads $\{CT, \tau, \sigma_t\}$ to the peer.

6.3 Data Storage Phase

We combine on-chain and off-chain to relieve storage pressure and protect data privacy.

6.3.1 Off-Chain Storage

After receiving a service request from the terminal, the peer queries the terminal registration information by mapping ide and then runs the $DS.Verify(\sigma_t, PP, did_i, pk_i, CT)$ algorithm to verify the legitimacy and integrity of CT. The off-chain storage provides a relational database and a file system to store CT, which the peer maintains. Remarkably, the CT will be broadcasted among peers through the transmission mechanism. Even if some peers go offline, the peers with copies of the data still provide data accessibility and effectively reduce the risk of a single point of failure. Meanwhile, multiple copies of data enable disaster recovery when data stored in the peer is lost.

File System: The file system stores unstructured data based on content identity (CID), where the CID is used to locate the data path $path_{cid}$. In addition, peers provide computational capabilities for integrity verification, such as constructing the MHT root of a dataset. The peer receives the encrypted unstructured data CT and calculates its CID. The $H(\cdot)$ algorithm computes the CIDs $cid_{blocks} = \{H(C_1), H(C_2), \ldots, H(C_n)\}$ of data blocks. The CID cid_{CT} of the CT is the MHT root constructed by cid_{blocks}. After the data is stored, the peer stores $path_{cid} = ch, cid_{CT}, cid_{blocks}$ as structured data in the relational database and submits its key information msg to an MQ waiting to be packaged.

Relational Database: The relational database stores structured data, the key information of unstructured data, and blockchain information. An example is shown in Fig. 5. A data table has six columns: (1) ID denotes the identifier of data computed using the $H(\cdot)$ algorithm; (2) $Data$ denotes the encrypted structured data CT or the key information of unstructured data $path_{cid}$; (3) $Type$ denotes the data type, where T_S denotes structured data and T_U denotes unstructured data; (4) $TxID$ denotes the identifier of tx that is the MHT root $H(S_{1,2,\ldots,i})$ of the MSG (see Fig. 6); (5) $PathProof$ consists of $Path$ and $Proof$, where $Path$ denotes the serial number sn of the data in data blocks and $Proof$ denotes sibling nodes to the MHT path of the data block; (6) $Height$ denotes the height of the block the transaction is in. When the peer receives the structured data submitted by terminals or key information of unstructured data submitted by peers, it creates a new record containing fields $< ID, Data, Type >$. After executing the packaging mechanism, the fields $< TxID, PathProof >$ in the record are updated. After the BCN successfully confirms the data, the field $< Height >$ is updated.

6.3.2 On-Chain Storage

The key information of off-chain data is stored on-chain, which protects data privacy and integrity. As can be seen from Fig. 5, the data's key information is the other fields other than $Data$, where the ID of structured data is $H(CT)$, and the ID of the unstructured data is $H(path_{cid})$. After the data packaging process (which will be introduced in Section 6.4), the packaged transaction tx is recorded in the BCN to guarantee the integrity of off-chain data.

ID	Data	Type	TxID	PathProof	Height
$H(CT_1)$	CT_1	T_{S_1}	$H(S_{1,2,\ldots,i_1})$	$[sn_1, path_{1,1}, \ldots, path_{1,log_2(i_1)+1}]$	$height_1$
$H(path_{cid_2})$	$path_{cid_2}$	T_{U_2}	$H(S_{1,2,\ldots,i_2})$	$[sn_2, path_{2,1}, \ldots, path_{2,log_2(i_2)+1}]$	$height_2$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
$H(CT_n)$	CT_n	T_{S_n}	$H(S_{1,2,\ldots,i_n})$	$[sn_n, path_{n,1}, \ldots, path_{n,log_2(i_n)+1}]$	$height_n$

Fig. 5. Data table structure.

6.4 Data Processing Phase

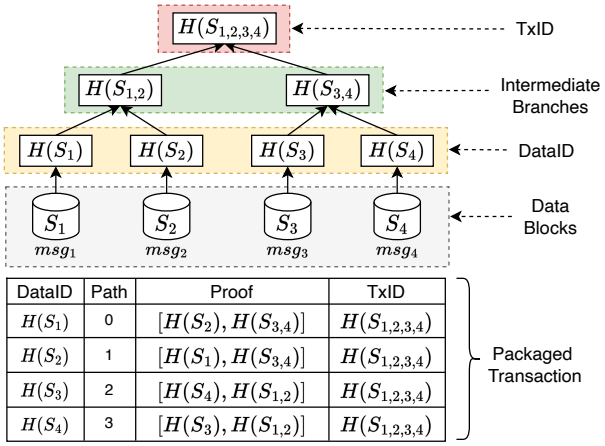
Many devices concurrently connect to the peer and submit many real-time service requests, which may affect system performance. Therefore, we propose an MHT-based packaging mechanism using the MQ. To reduce the communication overhead between the peer and the MQ, each peer maintains an MQ and belongs to the same subnet.

6.4.1 Packaging Policy

The packaging policy is configured in the system chain and is loaded by the peer when the channel is created. It can be set as follows: the number of messages in the MQ (e.g., 32), the total data size (e.g., 1MB), or the packaging timeout time (e.g., 2s). The manager can dynamically adjust the packaging policy according to the evolution of the business.

6.4.2 Packaging Mechanism

We propose an MHT-based packaging mechanism using the MQ to reduce the pressure of on-chain storage and improve system performance. After the peer stores the data off-chain, it pushes its key information msg to the MQ. Once the packaging policy is satisfied, the peer pulls a fixed amount i of messages $MSG = \{msg_1, msg_2, \ldots, msg_i\}$ from the MQ.



DataID	Path	Proof	TxID
$H(S_1)$	0	$[H(S_2), H(S_{3,4})]$	$H(S_{1,2,3,4})$
$H(S_2)$	1	$[H(S_1), H(S_{3,4})]$	$H(S_{1,2,3,4})$
$H(S_3)$	2	$[H(S_4), H(S_{1,2})]$	$H(S_{1,2,3,4})$
$H(S_4)$	3	$[H(S_3), H(S_{1,2})]$	$H(S_{1,2,3,4})$

Fig. 6. The packaging mechanism based on Merkle Hash Tree.

The steps to construct an MHT include that:

1) The S is used as data blocks msg to generate leaf nodes using the $H(\cdot)$ algorithm;
2) The two neighboring leaf nodes are hashed, and the result is used as the branch;
3) The two branches are hashed, and the procedure is repeated, with the last one being the root node.

Generally, we set the packaging policy to $i = 2^x, x \in N^*$ to decrease the number of hashes when building the MHT. Eventually, the depth of the constructed MHT is $log_2 i + 1$, i.e., $x + 1$. Without loss of generality, we assume that $x = 2$, and Fig. 6 shows the MHT structure and the packaged transaction constructed using 4 data blocks. The dataset is $S_{example} = \{S_1, S_2, S_3, S_4\}$, and the data is hashed to get the set of leaf nodes $S_{leaf} = \{H(S_1), H(S_2), H(S_3), H(S_4)\}$. We stitch $H(S_1)$ and $H(S_2)$ and hash it to get $H(S_{1,2})$, and $H(S_{3,4})$ is the same. Finally, the root node $S_{root} = H(S_{1,2,3,4})$ is obtained by hashing $H(S_{1,2})$ and $H(S_{3,4})$ after stitching. We package the $S_{example}$ in the MHT into a single transaction tx that contains the following: (1) $DataID$ denotes the identifier of the data; (2) $Path$ denotes the serial number of the data in data blocks; (3) $Proof$ denotes sibling nodes to the MHT path of the data block and can be used to verify the integrity of the data; and (4) $TxID$ denotes the identifier of the

tx, which is the MHT root. At last, the tx is broadcast to consensusers.

6.5 Data Sharing Phase

The collected data can be shared with others to enable data availability and break the information isolation. However, users will only feel secure and willing to share data if they have complete control over their data. We encrypt data using the CP-ABE, where the access structures implicit in the ciphertext determine who can use the data.

6.5.1 Access Policy

To ensure that only legitimate entities can access the shared data, the access policy $\mathbb{S}$ is part of the access structure $\mathbb{A}$, which is embedded in the ciphertext, and they are stored off-chain. The $\mathbb{S}$ is the set of attributes needed to decrypt the CT. Usually, the operator presets it inside the terminal. For example, only hospital doctors or the patient himself can access the patient's medical information. The access policy is expressed as $\mathbb{S} = \{(Hospital \& Doctor)|(PatientHimself)\}$, where $\&$ denotes the AND logic and $|$ denotes the OR logic.

6.5.2 Encrypt

An efficient encryption algorithm is more beneficial for sharing data $DATA_s$. The AES symmetric encryption algorithm is more efficient than the public key encryption algorithm. Moreover, encryption security is also essential for data sharing, and the key of the AES algorithm is generated using a one-time pad (OTP) algorithm. For the divided unstructured data, the same key is used for all the data blocks. The $\mathbb{A}$ is constructed by the $\mathbb{S}$ and the key, and only entities satisfying the access policy can resolve key. The terminal invokes the $DS.Encrypt(PP, DATA_s, \mathbb{A})$ algorithm obtains the CT, and it uploads CT to the peer, which occurs during the data transmission phase (see Section 6.2).

6.5.3 Decrypt

The authorized entity performs the decryption operation, and the peer cannot view the decrypted data. Before decrypting the shared data, the entity sends its set of attributes $\mathbb{S}$ to the peer. The peer runs the $DS.GenDecryptKey(ssk, \mathbb{S})$ algorithm to compute the key and returns it to the entity. Then, the entity sends a view request for the data by uploading the data identifier $DataID$ to the peer. The peer queries the CT based on the $DataID$ and returns it to the entity over a secure channel. The entity invokes the $DS.Decrypt(PP, CT, key)$ algorithm to decrypt the CT and obtain the shared data. Only the entity that satisfies the data access structure can decrypt the CT, for example, an entity with the attributes $\mathbb{S}_{True} = \{Hospital, Doctor\}$. However, an entity that has the attribute $\mathbb{S}_{False} = \{Hospital, Nurse\}$ cannot decrypt it.

6.6 Data Auditing Phase

The peers have the supreme permission to manage off-chain data, and it is essential to protect the integrity and availability of the data. After the BCN confirms the uploaded data, the terminal can immediately invoke an auditing request to verify the availability of the data. The terminals can also periodically invoke an auditing request to verify the integrity of the data.

6.6.1 Auditing Policy

An auditing policy is defined in configuration files and can be dynamically updated on the system chain. The auditing operation can be set to be executed quantitatively (e.g., 500 transactions) or periodically (e.g., once a day).

6.6.2 Auditing Procedure

The terminals initiate an auditing request running the $DS.GenChal(pk_i, did_i)$ algorithm generates the challenge $chal$ and submits it to the arbitrary peer. When the peer receives an auditing request, it runs the $DS.ProofGen(did_i, chal, pk_i, cid_i)$ algorithm to obtain $\{R_{peer}, R_{bc}\}$. The R_{bc} represents the MHT constructed by the arbitrary peer (public auditor with blockchain query capability) through the hashtag of on-chain data. The R_{peer} represents the MHT generated by the audited peer (who stores the copy of the data) based on off-chain data. Remarkably, the R_{peer} obtainment depends on the data type being audited. The audited peer constructs an MHT for structured data by querying the local relational database. For unstructured data, it queries the file system. Finally, the terminal runs the $DS.VerifyProof(R_{peer}, R_{bc})$ algorithm to verify the integrity of data and submits the result to the BCN. The terminal can also immediately query the key information of the on-chain data after the BCN confirms the data. By comparing the results with local computations, the terminal verifies the availability of the on-chain and off-chain data.

7 Implementation

We implement the DSChain using Golang (v1.18.5) with approximately 45,000 lines of code. Specifically, we implement our integration scheme with the GoWrapper [2], which is a port of the pairing-based cryptography (PBC) library (v0.5.14) [3]. We implement the scheme on Type A curve $y^2 = x^3 + x$ over the field F_q, where the order p is 160 bits and the group size q is 512. The hash function $H(\cdot)$ is instantiated with the SHA256 hash algorithm, and we use the first 160 bits of the output; The hash function $h(\cdot)$ is realized by using $H(\cdot)$ to obtain an element $h' \in \mathbb{Z}_p^*$.

Our system is a blockchain with a permission mechanism and chain structure. We use the open-source GoFastDFS[4] as a file system to store unstructured data and MySQL as a relational database to store structured data. We choose LevelDB as a Key-Value database to index the current state and transactions. We use RabbitMQ as an MQ and queue information to distinguish the data type.

All nodes form the P2P network and classify the peer affiliation according to the channel. The transmission mechanism uses the Gossip protocol [39], which is a decentralized and distributed protocol. The messages are synchronized among the peers, so it is the ultimate consistency protocol.

2. https://github.com/Nik-U/pbc
3. https://crypto.stanford.edu/pbc/
4. https://github.com/sjqzhang/go-fastdfs

There are two verification mechanisms in our system. The first is to verify the identity using a self-signed SSL/TLS certificate issued by the manager. The other one is to use CLS keys generated by peers, which is one-way verification.

Our system supports pluggable consensus protocols that can be replaced according to different cases. To facilitate testing, we adopt the Solo consensus mechanism, which uses RabbitMQ maintained by consensusers as the sequencing container for transaction consistency.

We use lightweight and portable Docker containers to provide execution environments for smart contracts. Our system can be extended by programming a concrete operation logic using Golang. The interface to the smart contract infrastructure provides two operations:

- $init(stub)$: The system will inversely call all implementation logic of this interface when loading smart contracts, which will only be executed once.
- $invoke(stub)$: The interface provides read-write operations on the data within the smart contract and allows users to design the execution logic within the system's constraints.

The $stub$ in the above interfaces encapsulates the parameters submitted by the invoker and the system information. Eventually, they will return the execution result to the invoker.

The configuration file specifies system parameters, loaded after the system boots and uploaded to the system chain. The system manager can dynamically update the configuration, and the latest block specifies the current configuration on the system chain.

8 Evaluation and Analysis

This section evaluates the overhead of cryptographic protocols and system performance. Then, we analyze the security of the system and the integration scheme.

8.1 Environment Setup

We developed an automated testing tool using Golang (v1.18.5) to help us evaluate our system. It allows setting the sending rate and the duration of testing. Furthermore, it runs ten rounds by default to reduce occasionality. In the network, the loss of communication messages is inevitable, so we set the tolerable failure rate (FR) to $1‰$, which means that one out of every 1,000 requests is allowed without a response. In addition, considering the impact of response time on user experience, we discard timeout (i.e., 3 seconds) requests. Also, the system may fluctuate during runtime, and we use the 5% error line to mark the average latency.

We use universal measurements to evaluate system performance, which includes system throughput and transaction latency. The latency indicates the time it takes for a message to start from the terminal request to be confirmed by the BCN. The typical definition of throughput: $X = \frac{C}{\tau}$, where C denotes the number of completed transactions, and τ denotes the time it takes to complete these transactions.

Without knowing the actual throughput, we first set the sending rate in 100 tps step size to find a valid range for the optimal throughput. Then, it is gradually approximated within that range using a dichotomy. Finally, the certainty of data occurrence is verified multiple times based on the optimal throughput. Unless otherwise stated, we measure the optimal performance with zero FR.

Unless explicitly mentioned differently in our experiments: (1) All measurements are obtained from the testing tool mentioned above, which runs on a terminal node; (2) Nodes are hosted in AliCloud Elastic Compute Service (ECS) and communicate with each other over 0.8 Gbps of intranet bandwidth and a private network; (3) All nodes run on Intel Xeon Platinum 8269CY 4-vCPU @ 2.50GHz virtual machines (VMs) with Ubuntu 20.04 (64-bit) operating system, 8 GB of RAM, and 20G of SSD (with 1,960 IOPS) as a local disk; (4) A single node runs the Solo consensus service, a channel contains one peer and one node emulates the terminal, all on distinct VMs. Nodes communicate securely using the SSL/TLS protocol. The node clocks are synchronized with the AliCloud Network Time Protocol (NTP) service during the experiment.

8.2 Performance Evaluation

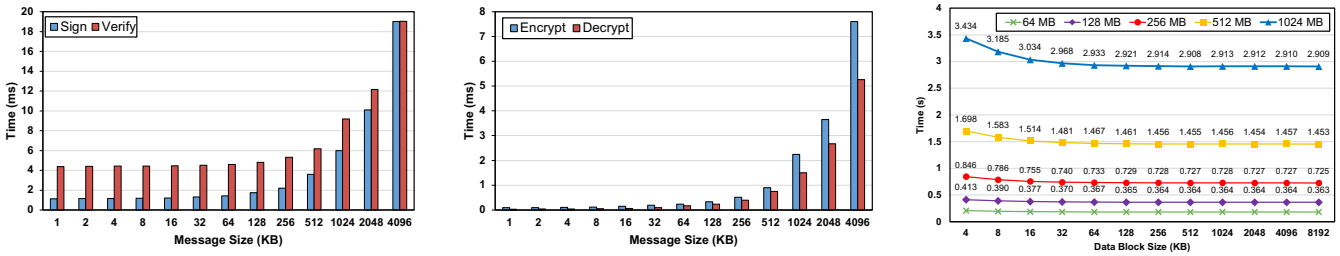
The system's performance determines what it can do, enabling analysis of whether it can support large-scale IoT applications. We measure the overhead of cryptographic protocols and then analyze the impact of different parameters on performance. Finally, we conduct a performance comparison between DSChain and Fabric to demonstrate the availability of our system.

8.2.1 Cryptographic Protocols Evaluation

To measure the impact of cryptographic protocols overhead on transaction latency, we use the Benchmark testing tool provided by Golang to evaluate the operations overhead in the CLS, CP-ABE, and PA phases. We use the VM with 4-vCPU and 8 GB of RAM mentioned above. The experimental results are shown in Fig. 7.

During data processing, message size may affect the system's processing rate. Without loss of generality, we increase the message size from 1 KB to 4,096 KB for evaluation. From Fig. 7a and Fig. 7b, when the message size is in the range of 1 KB and 512 KB, the time overhead of signature, verification, encryption, and decryption is relatively stable. After 512 KB, their time overhead begins to increase significantly. Since encrypting and signing messages occur together, we consider the message size 512 KB more suitable for transmission. The verification of messages by peers directly affects the throughput, and at 512 KB, the verification overhead is small. Therefore, the unstructured data can be divided into 512 KB blocks and uploaded in parallel to achieve better performance. In Fig. 7b, the time overhead of encryption is higher than decryption. The reason is that there is some time consumption in constructing the access structure during encryption. However, it occurs at the terminal, and the impact on system performance is negligible.

When generating data hashtags by the terminal and auditing by peers, constructing an MHT is a rather time-consuming operation. We measure the time overhead of constructing MHTs for message sizes of 64 MB, 128 MB, 256 MB, 512 MB, and 1,024 MB, respectively. Also, we increase the data block size after splitting from 4 KB to 8,192 KB. The experimental results are shown in Fig. 7c. We know



(a) Overhead of sign and verify. (b) Overhead of encryption and decryption. (c) Overhead of building Merkle Hash Tree.

Fig. 7. Overhead of cryptographic protocols.

that for a given data size, the data block size is inversely proportional to the number of blocks after splitting. The overhead decreases significantly when the data block size is less than 64 KB. We analyze that too many data blocks decrease rapidly, bringing the result. Remarkably, they all show a dent at 512 KB, meaning splitting the file into 512 KB has less overhead than 256 KB and 1,024 KB. A larger data block size and many blocks cause high time overhead, and 512 KB is the optimal balance. Therefore, it is reasonable to believe that 512 KB is a more suitable value for a data block and that larger blocks are inconvenient for transfer.

Analyzing the above experiments, 512 KB is a significant value due to its best overall performance in signature, verification, encryption, decryption, and construction of MHTs. We suggest limiting the size of the data uploaded by the terminal to around 512 KB to achieve optimal system performance.

8.2.2 System Performance Evaluation

We evaluate the impact of different factors on system performance as detailed below.

Impact of block size on performance: Block size is a critical factor that affects throughput and transaction latency. To determine the required block size for subsequent experiments, we increase the block size from 1 MB to 8 MB. The experimental results are shown in Fig. 8a. Throughput does not significantly increase after 6 MB, and the increased latency is within acceptable limits. Moreover, the FR is no longer zero when the block size exceeds 6 MB, which is 0.43‰at 7 MB and 0.89‰at 8 MB, respectively. The 6 MB block contains a large number of transactions, which is limited by the processing capacity of the consensus service nodes. Once they cannot be processed promptly, the consensus service will actively discard them, resulting in the FR. Therefore, we adopt 6 MB as the block size for subsequent experiments.

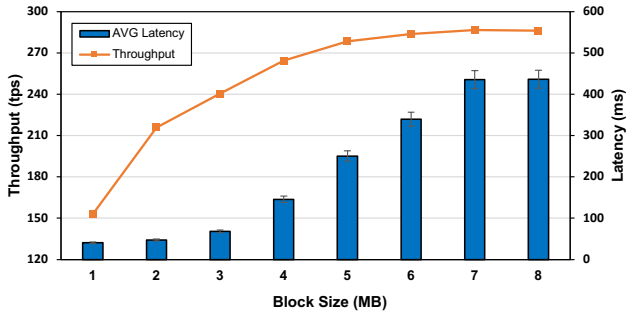
Impact of packaging mechanism on performance: The peers pre-process data by building an MHT before packaging them. To measure the impact of the number of data blocks in the MHT on throughput and latency, we set the number of data blocks by increasing exponentially with a base of 2. Based on the previous experiment, we adopt the block size of 6 MB and keep the same experimental settings. We use three VMs running the consensus service, the peer, and the terminal. The experimental results are shown in Fig. 8b.

As can be seen, with data blocks from 2 to 32, the throughput increases significantly with the increase of data blocks in a linear correlation and reaches 1,035 tps at 32. After 32 data blocks, the throughput no longer improves significantly, accompanied by increased latency. When increasing to 256, the throughput increases to 1,080 tps, the latency rises to 804 ms, and the FR begins to appear. We are more interested in 32 data blocks. The above results may be related to the processing power of a single peer. It causes some threads to be stranded within the system runtime when exceeding 32 data blocks during the data packaging process, which affects performance. In conclusion, after the packaging process, the throughput increases from the original 280 tps to 1,080 tps, demonstrating that our packaging mechanism can effectively improve performance.

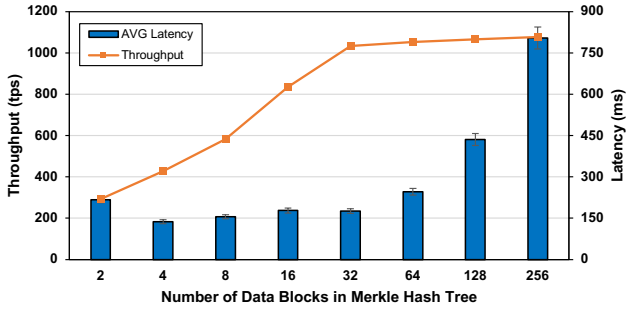
Impact of peer CPU and RAM on performance: To analyze the optimal hardware facility for the system to be stable running, we attempt to change the number of CPU cores (vCPUs) and RAM size of the peer node. We set it with 4-vCPU and 8 GB of RAM as the baseline for comparison. The experimental results are shown in Fig. 8c. When the number of vCPUs increases to 8-vCPU, its performance varies slightly. Then, we fix the vCPUs to 4-vCPU and increase the RAM to 16 GB, which still has a slight variation. The results indicate that increasing the hardware facility does not effectively improve performance.

We know that the baseline setting is sufficient for the system to achieve optimal performance, and then we try out the minimum hardware facility for the system to stable running. We degrade the hardware facility from the baseline to 2-vCPU and 4GB of RAM, respectively, and see that the throughput decreases from 1,035 tps to 1,010 tps, and the latency is almost unchanged. Furthermore, we continue to degrade the hardware facility to 2-vCPU and 4GB of RAM and find that the throughput only drops by ten tps. In other words, our system performs well enough with the hardware facility of 2-vCPU and 4GB of RAM. We intend to continue to degrade to clarify the impact of RAM size and vCPUs on performance.

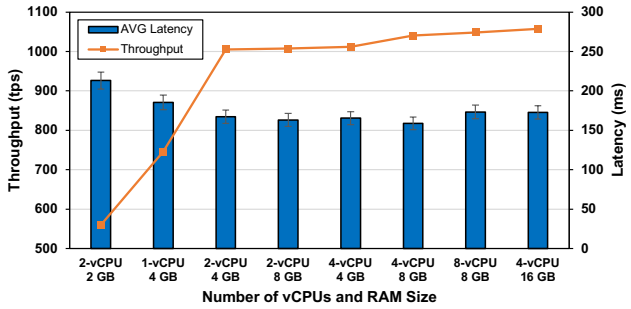
First, we degrade the hardware facility to 1-vCPU and 4GB of RAM. At this point, a noticeable reduction in throughput occurs (approximately 750 tps). We analyze that fewer vCPUs are unable to process intensive terminal requests. Next, we continue to degrade the hardware facility to 2-vCPUs and 2GB of RAM, and the throughput decreases to approximately 560 tps. The RAM size has a greater impact on performance than vCPUs. We speculate that the peer



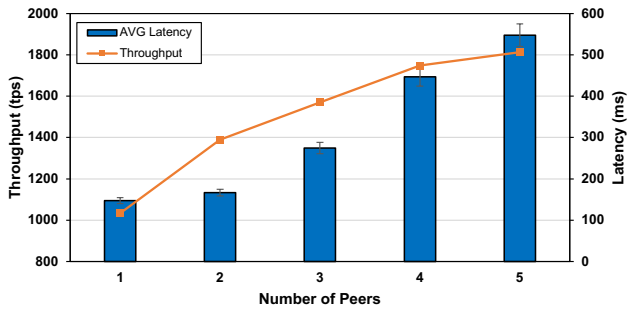
(a) Impact of block size on throughput and latency.



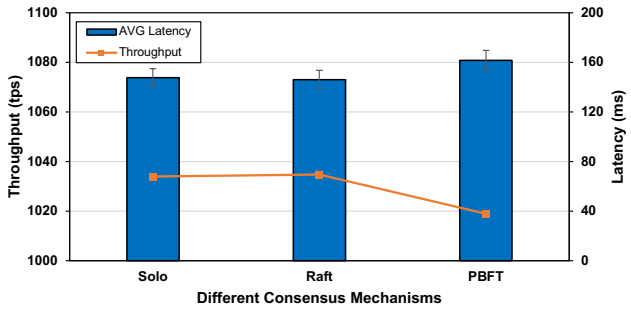
(b) Impact of the number of data blocks on throughput and latency.



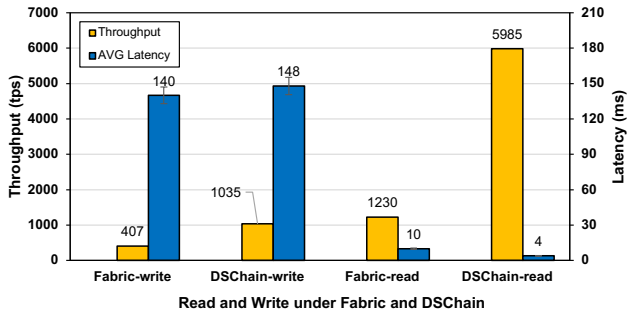
(c) Impact of peer CPU and RAM on throughput and latency.



(d) Impact of the number of peers on throughput and latency.



(e) Impact of different consensus algorithms on system performance.



(f) Performance comparison between DSChain and Fabric.

Fig. 8. Impact of different parameters on throughput and latency.

needs to cache data while waiting for packaging, which consumes RAM. Moreover, there is a brief RAM outage and system instability when processing requests intensively. To ensure system stability, we recommend running the peer with a 4-vCPU and 8 GB of RAM, although running it with a 2-vCPU and 4 GB of RAM is possible.

Scalability: To better apply our system to large-scale IoT applications, we evaluate its scalability by increasing the number of peers from 1 to 5. The experimental results are shown in Fig. 8d. We increase to 2 peers, the throughput reaches approximately 1,390 tps, and the latency shows only slight variation, which is negligible. We continue to increase to 3 peers, the throughput reaches approximately 1,570 tps, and the latency rises to 274 ms, which has little effect on the system. When we increase to 4 and 5 peers, the throughput reaches 1,750 tps and 1,810 tps, respectively, and the latency rises to 447 ms and 547 ms, respectively. Notably, while throughput increases, the rising rate decreases, and the latency increases severely.

The throughput is linearly related to the number of peers but not in a simple multiplicative way. We know that peers and consensusers process the data from terminal submission to confirmation. Multiple peers can achieve parallel processing of most operations. However, a single consensuser may reach the processing bottleneck. While increasing the number of peers, increasing the corresponding number of consensus nodes is also necessary to ensure the system's stability. Meanwhile, the experimental results demonstrate that our system is scalable.

Impact of different consensus algorithms on system performance: In Experiments 1–4, we used the Solo consensus service provided by a single node to facilitate deployment and testing. To verify the impact of different consensus mechanisms on system performance and clarify the consensus layer's pluggability, we develop three typical consensus mechanisms: Solo, Raft, and PBFT. Remarkably, Raft as a crash fault-tolerant consensus protocol can tolerate $(n-1)/2$ crashed nodes, PBFT as a Byzantine fault-tolerant

consensus protocol can tolerate $(n-1)/3$ evil nodes, where n denotes the total number of nodes. The experimental results are shown in Fig. 8e.

As shown in the previous experiments, the throughput and latency are approximately 1,035 tps and 148 ms, respectively, when using the Solo consensus mechanism with 6 MB block size and 32 data blocks. When switching to the Raft consensus mechanism, the throughput and latency are approximately 1,030 tps and 147 ms, and with PBFT, are about 1,020 tps and 162 ms, respectively. These consensus mechanisms have relatively little impact on performance.

We analyze that the above observation is related to the workflow of consensus mechanisms. The Solo consensus mechanism runs on a single node, sequences and packages messages from peers according to a predefined policy, and delivers blocks to the corresponding peers. The Raft consensus mechanism consists of three roles: leader, candidate, and follower. The leader receives messages from peers and replicates logs to followers after consensus. If the leader node is faulty, a new leader is elected after a heartbeat timeout, and the impact on latency depends on the timeout interval. Therefore, the Raft consensus mechanism is mainly provided by the leader, while the replication of logs and a leader election have little impact on performance. The PBFT consensus mechanism is provided by a master node and multiple slave nodes and consists of three phases: pre-prepare, prepare, and commit. Increased communication overhead leads to a higher latency and a lower throughput.

In summary, the experimental results demonstrate that our system has pluggability and that the impact of Raft and PBFT consensus mechanisms on system performance is acceptable. Therefore, managers can choose different consensus mechanisms to adapt to businesses.

8.2.3 Performance Comparison With Fabric

Our system is a blockchain with a permission mechanism that secures the complete data lifecycle. To validate the effectiveness of the proposed system, we conduct a performance comparison between DSChain and Fabric. Fabric is one of the most popular blockchain systems that supports enterprise-level applications. It has access control, data isolation, and scalability advantages, which are highly similar to our system. Our system runs on three VMs with 4-vCPU and 8GB of RAM, i.e., the consensus service, the peer, and the terminal are on distinct VMs. Accordingly, Fabric uses the same hardware facilities. They use the Solo consensus mechanism. To more comprehensively compare the differences between the two systems, we measure the performance by reading (querying) and writing (invoking) the transactions. The experimental results are shown in Fig. 8f.

Blockchain as a distributed database, the performance of write operations can more intuitively reflect its availability. The throughput and latency of DSChain in invoking transactions are approximately 1,035 tps and 148 ms, respectively, while Fabric is about 407 tps and 140 ms, respectively. The transaction latency of our system is 8 ms more than Fabric. We use some computational overhead on the cryptographic protocols (see Section 8.2.1) to secure the complete data lifecycle. However, the throughput of our system is about 2.5 times that of Fabric, benefitting from the MHT-based packaging mechanism we proposed. The query response of a transaction is also an important factor in measuring the performance. The throughput and latency of DSChain in querying transactions are approximately 5,985 tps and 4 ms, respectively, while Fabric is about 1,230 tps and 10 ms, respectively. The throughput of our system is approximately 4.9 times that of Fabric and has lower latency. It benefits from the off-chain storage capability of our system, where peers directly query relational databases and file systems without parsing on-chain blocks. Overall, the performance of our system is better than Fabric. Hence, it can better process transactions in real-time and be used for large-scale IoT applications.

8.3 Security Analysis

In conjunction with the blockchain prototype system and the integration scheme, we provide a security analysis of the blockchain network and the integration scheme.

8.3.1 Blockchain Security

DSChain is a decentralized distributed blockchain system, and we assume that attackers can launch the following attacks. We are not concerned about how attackers launch different attacks but focus on defending the system against these possible attacks.

Man-in-the-Middle (MITM) Attack: Our system is protected against MITM attacks using a public key cryptosystem. We know that after the terminal registers its identity, the peer only retains the partial private key, and the terminal generates the public-private key pair locally. The terminal keeps the private key in a secure key container, and the public key is published to the BCN, guaranteeing the transmitted data's security by CLS. Since the attackers in an MITM attack do not have the private key of the terminal, they cannot obtain any plaintext information about the terminal. They cannot tamper with the data during transmission. Therefore, no one (not even a peer) can impersonate the identity of the terminal.

Distributed Denial of Service (DDoS) Attack: Data is redundantly replicated among all peers in the channel, resilient to the failure of one or more nodes, increasing our system's availability. Also, the immutability of blockchain can be used to manage IoT devices and their data. The peers can check the registration information of terminals to verify all service requests. Denial of service to unauthorized IoT devices can effectively prevent DDoS attacks.

8.3.2 Security Statement

As can be seen in Section 5, our integration scheme is constructed by integrating the CLS, CP-ABE, and PA paradigms. We consider differences and commonalities in the design details of instances under those paradigms and that the paradigms can standardize the instances. In other words, arbitrary instances that satisfy our defined paradigm can be integrated into our system. In our integration scheme, given a security parameter λ, the $DS.SystemInit$ algorithm outputs the public parameters PP. Since PP are the common parameters for CLS, CP-ABE, and PA, it is strongly required that instances of all paradigms use the same security level of λ. In addition, our integration scheme keeps the

structure of the defined paradigms intact. The threat models of all paradigms are maintained unchanged. The specific instance must satisfy the definition of the paradigm. Therefore, the instance's security will support itself, which is not our primary concern. For example, in the CLS instance [36], Jia et al. proposed that there may be a fake KGC threat and a public-key replacement attack and provide detailed security proof. Jia et al.'s scheme is consistent with the CLS paradigm, and both Waters et al.'s [37] and Li et al.'s schemes [38] are also. Due to the limited space, the security proofs for each instance will not be explained in detail here.

8.3.3 Security Objectives Analysis

We analyzed the security objectives from a system perspective, as detailed below.

Trustworthiness: The CLS paradigm requires terminals to submit trusted DIDs for registration, and only authorized terminals can initiate service requests to peers. When a malicious terminal threatens the BCN, peers can track the terminal through its registration information. The traceability of CLS can regulate the behavior of the terminals.

Confidentiality: In the CP-ABE paradigm, the ciphertext is identified with access structure and the private key with attributes. If the entities are not authorized, they cannot view the information. The CP-ABE ensures that only if the entity's attributes match the decryption conditions can decrypt the ciphertext. We also use the AES symmetric encryption algorithm with higher encryption efficiency as the secret value for CP-ABE, and its security is widely recognized. Also, to prevent entities from decrypting other files by calculating the secret value, we use the OTP algorithm to ensure that the relationship between data and key is one-to-one. Even if the entity gets the key for that data, it cannot decrypt others, ensuring the confidentiality of the data.

Privacy: We use a combination of on-chain and off-chain to store encrypted data, which prevents attackers from analyzing user behavior through ciphertext on the BCN. The encrypted data is stored off-chain, and its key information is stored on-chain, which ensures that the ciphertext is invisible. Meanwhile, the channel isolation technology allows only entities within the channel to view the key information.

Integrality: A periodic audit of off-chain data can regulate the behavior of peers. The off-chain stores encrypted data and corresponding hashtags, and the validation path and key information of data blocks are stored on-chain. MHT has natural properties regarding fast integrality validation, and corrupted data will not pass verification.

9 Related Work

Recently, the rapid development of blockchain technology has dramatically impacted politics, economics, culture, and our lives. Melanie Swan divides the development stages of blockchain into blockchain 1.0, blockchain 2.0, and blockchain 3.0 [3]. The emergence of Bitcoin [2] marked the arrival of the blockchain 1.0 era, which addressed the issues of cryptocurrency and decentralization. Blockchain 2.0, represented by Ethereum [11], solved the market's decentralization problem, and smart contracts emerged. Blockchain 3.0 is the era of comprehensive blockchain applications, which can build applications in various fields.

Existing blockchains are divided into three types according to different degrees of openness: private, public, and consortium. Bitcoin and Ethereum represent the public blockchain, the consortium blockchain is represented by Hyperledger Fabric [12], and the private blockchain is represented by AntChain[5] proposed by Ant Group. Block organization structure is divided into chain structure blockchain and directed acyclic graph (DAG) structure blockchain. Existing blockchains are mainly based on chain structure, such as Bitcoin, Ethereum, and Quorum in order-execute mode and Hyperledger Fabric in execute-order-validate mode. To improve system performance, blockchains with a DAG architecture emerge, such as IOTA [16] and XDAG[6].

All peers in the P2P network communicate using propagation protocols such as Gossip [39] and TeleHash. Some blockchain systems extend system capabilities by providing the ability to execute scripts. For example, Bitcoin provides a scripting language based on stack execution, Ethereum provides smart contracts based on the Ethereum virtual machine (EVM), and Fabric provides chaincodes based on Docker containers. Consensus mechanisms are key technologies to ensure transaction consistency, such as proof of work (PoW), proof of stake (PoS), delegated proof of stake (DPoS), Raft [34] and PBFT [35]. Cryptographic protocols support the tamper-evident and data-secure features of blockchain. Cryptographic protocols are mainly divided into symmetric encryption algorithms (e.g., AES, DES, 3DES), asymmetric encryption algorithms (e.g., RSA, DSA, ECC), and hash algorithms (e.g., MD5, SHA1, HMAC, SHA256), where symmetric encryption algorithms are generally more efficient than asymmetric encryption algorithms. Elliptic curve cryptography (ECC) and RSA are frequently used asymmetric encryption algorithms, where ECC can use shorter keys to achieve similar or higher security than RSA.

10 Conclusion

There are many data security challenges in IoT, and we propose DSChain, a blockchain system that can protect the complete lifecycle security of data. We integrate a CLS paradigm to ensure a trusted source of collected data, a CP-ABE paradigm to achieve fine-grained access control for shared data, and a PA paradigm to ensure the integrity and availability of stored data. Meanwhile, we combine on-chain and off-chain to store massive data, reducing storage pressure and protecting user privacy. In addition, we propose an MHT-based packaging mechanism to package multiple messages into a single transaction to improve system performance. We fully implement DSChain and evaluate its performance regarding throughput and latency. The experimental results indicate that DSChain can achieve approximately 1,035 tps on a single peer and less than 200 ms latency. Furthermore, our system is scalable. Managers can adjust the network scale according to requirements.

DSChain combines on-chain and off-chain storage to reduce the storage pressure of the blockchain. However, the data generated by IoT terminals will be immeasurable.

5. https://antchain.antgroup.com
6. https://xdag.io

Our future work will focus on data security solutions and optimizing data storage, for example, using data hierarchical storage and deduplication techniques to optimize unnecessary data storage.

REFERENCES

[1] R. El Sibai, N. Gemayel, J. Bou Abdo, and J. Demerjian, "A survey on access control mechanisms for cloud computing," *Transactions on Emerging Telecommunications Technologies*, vol. 31, no. 2, p. e3720, 2020.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[3] M. Swan, *Blockchain: Blueprint for a new economy*. O'Reilly Media, Inc., 2015.

[4] Z. Xiong, Y. Zhang, N. C. Luong, D. Niyato, P. Wang, and N. Guizani, "The best of both worlds: A general architecture for data management in blockchain-enabled internet-of-things," *IEEE Network*, vol. 34, no. 1, pp. 166–173, 2020.

[5] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with iot. challenges and opportunities," *Future generation computer systems*, vol. 88, pp. 173–190, 2018.

[6] F. Daidone, B. Carminati, and E. Ferrari, "Blockchain-based privacy enforcement in the iot domain," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 3887–3898, 2021.

[7] A. Al Sadawi, M. S. Hassan, and M. Ndiaye, "A review on the integration of blockchain and iot," in *2020 International conference on communications, signal processing, and their applications (ICCSPA)*. IEEE, 2021, pp. 1–6.

[8] Y. Li, W. Susilo, G. Yang, Y. Yu, D. Liu, X. Du, and M. Guizani, "A blockchain-based self-tallying voting protocol in decentralized iot," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 119–130, 2020.

[9] L. Da Xu, Y. Lu, and L. Li, "Embedding blockchain technology into iot for security: A survey," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 452–10 473, 2021.

[10] D. E. Kouicem, Y. Imine, A. Bouabdallah, and H. Lakhlef, "Decentralized blockchain-based trust management protocol for the internet of things," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 1292–1306, 2020.

[11] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[12] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[13] G. Jeong, N. Lee, J. Kim, and H. Oh, "Azeroth: Auditable zero-knowledge transactions in smart contracts," *IEEE Access*, pp. 1–1, 2023.

[14] T. Waltonchain. Waltonchain white paper. 2022, Oct 12. [Online]. Available: https://www.waltonchain.org/#/whitePaper

[15] T. IoTeX. Iotex: a decentralized network for internet of things powered by a privacy-centric blockchain. [Online]. Available: https://v1.iotex.io/white-paper

[16] W. F. Silvano and R. Marcelino, "Iota tangle: A cryptocurrency to communicate internet-of-things data," *Future Generation Computer Systems*, vol. 112, p. 307–319, 11 2020.

[17] B. Farahani, F. Firouzi, and M. Luecking, "The convergence of iot and distributed ledger technologies (dlt): Opportunities, challenges, and solutions," *Journal of Network and Computer Applications*, vol. 177, p. 102936, 2021.

[18] A. Connolly, J. Deschamps, P. Lafourcade, and O. Perez Kempner, "Protego: Efficient, revocable and auditable anonymous credentials with applications to hyperledger fabric," in *International Conference on Cryptology in India*. Springer, 2022, pp. 249–271.

[19] Z. Zheng, T. Wang, A. K. Bashir, M. Alazab, S. Mumtaz, and X. Wang, "A decentralized mechanism based on differential privacy for privacy-preserving computation in smart grid," *IEEE Transactions on Computers*, vol. 71, no. 11, pp. 2915–2926, 2021.

[20] Y. Lin, J. Li, S. Kimura, Y. Yang, Y. Ji, and Y. Cao, "Consortium blockchain-based public integrity verification in cloud storage for iot," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3978–3987, 2021.

[21] T. Meng, Y. Zhao, K. Wolter, and C.-Z. Xu, "On consortium blockchain consistency: A queueing network model approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 6, pp. 1369–1382, 2021.

[22] S. Brotsis, K. Limniotis, G. Bendiab, N. Kolokotronis, and S. Shiaeles, "On the suitability of blockchain platforms for iot applications: Architectures, security, privacy, and performance," *Computer Networks*, vol. 191, p. 108005, 2021.

[23] Y. Liu, C. Zhang, Y. Yan, X. Zhou, Z. Tian, and J. Zhang, "A semi-centralized trust management model based on blockchain for data exchange in iot system," *IEEE Transactions on Services Computing*, 2022.

[24] E. Samir, H. Wu, M. Azab, C. Xin, and Q. Zhang, "Dt-ssim: A decentralized trustworthy self-sovereign identity management framework," *IEEE Internet of Things Journal*, vol. 9, no. 11, pp. 7972–7988, 2021.

[25] J. Lu, J. Shen, P. Vijayakumar, and B. B. Gupta, "Blockchain-based secure data storage protocol for sensors in the industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 8, pp. 5422–5431, 2021.

[26] K. Miyachi and T. K. Mackey, "hocbs: A privacy-preserving blockchain framework for healthcare data leveraging an on-chain and off-chain system design," *Information processing & management*, vol. 58, no. 3, p. 102535, 2021.

[27] Y. Du, H. Duan, A. Zhou, C. Wang, M. H. Au, and Q. Wang, "Enabling secure and efficient decentralized storage auditing with blockchain," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3038–3054, 2021.

[28] D. Liu, C. Huang, J. Ni, X. Lin, and X. S. Shen, "Blockchain-cloud transparent data marketing: Consortium management and fairness," *IEEE Transactions on Computers*, vol. 71, no. 12, pp. 3322–3335, 2022.

[29] J. Huang, L. Kong, J. Wang, G. Chen, J. Gao, G. Huang, and M. K. Khan, "Secure data sharing over vehicular networks based on multi-sharding blockchain," *ACM Transactions on Sensor Networks*, 2023.

[30] H. Hou, J. Ning, Y. Zhao, and R. H. Deng, "Fine-grained and controllably editable data sharing with accountability in cloud storage," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3448–3463, 2021.

[31] Z. Wan, W. Liu, and H. Cui, "Hibechain: A hierarchical identity-based blockchain system for large-scale iot," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[32] H. Kang, T. Dai, N. Jean-Louis, S. Tao, and X. Gu, "Fabzk: Supporting privacy-preserving, auditable smart contracts in hyperledger fabric," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019, pp. 543–555.

[33] I. Makhdoom, I. Zhou, M. Abolhasan, J. Lipman, and W. Ni, "Privysharing: A blockchain-based framework for privacy-preserving and secure data sharing in smart cities," *Computers & Security*, vol. 88, p. 101653, 2020.

[34] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 305–319. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

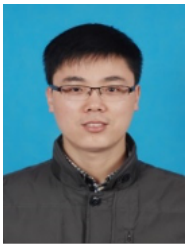
[35] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002.

[36] X. Jia, D. He, Q. Liu, and K.-K. R. Choo, "An efficient provably-secure certificateless signature scheme for internet-of-things deployment," *Ad Hoc Networks*, vol. 71, pp. 78–87, 2018.

[37] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *International workshop on public key cryptography*. Springer, 2011, pp. 53–70.

[38] J. Li, J. Wu, G. Jiang, and T. Srikanthan, "Blockchain-based public auditing for big data in cloud storage," *Information Processing & Management*, vol. 57, no. 6, p. 102382, 2020.

[39] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, "Peer-to-peer membership management for gossip-based protocols," *IEEE Transactions on Computers*, vol. 52, no. 2, pp. 139–149, 2003.



Jie Cui was born in Henan Province, China, in 1980. He received his Ph.D. degree in University of Science and Technology of China in 2012. He is currently a professor and Ph.D. supervisor of the School of Computer Science and Technology at Anhui University. His current research interests include applied cryptography, IoT security, vehicular ad hoc network, cloud computing security and software-defined networking (SDN). He has over 150 scientific publications in reputable journals (e.g. IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Information Forensics and Security, IEEE Journal on Selected Areas in Communications, IEEE Transactions on Mobile Computing, IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Vehicular Technology, IEEE Transactions on Intelligent Transportation Systems, IEEE Transactions on Network and Service Management, IEEE Transactions on Industrial Informatics, IEEE Transactions on Industrial Electronics, IEEE Transactions on Cloud Computing and IEEE Transactions on Multimedia), academic books and international conferences.

Yatao Li is now a research student at the School of Computer Science and Technology, Anhui University. His research focuses on the security of blockchain.

Qingyang Zhang was born in Anhui Province, China, in 1992. He received his B. Eng. degree and Ph.D. degree in computer science from Anhui University in 2021. He is currently a lecture of School of Computer Science and Technology at Anhui University. His research interest includes edge computing, computer systems, and security.

Hong Zhong was born in Anhui Province, China, in 1965. She received her PhD degree in computer science from University of Science and Technology of China in 2005. She is currently a professor and Ph.D. supervisor of the School of Computer Science and Technology at Anhui University. Her research interests include applied cryptography, IoT security, vehicular ad hoc network, cloud computing security and software-defined networking (SDN). She has over 200 scientific publications in reputable journals (e.g. IEEE Journal on Selected Areas in Communications, IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Mobile Computing, IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Information Forensics and Security, IEEE Transactions on Intelligent Transportation Systems, IEEE Transactions on Multimedia, IEEE Transactions on Vehicular Technology, IEEE Transactions on Network and Service Management, IEEE Transactions on Cloud Computing, IEEE Transactions on Industrial Informatics, IEEE Transactions on Industrial Electronics and IEEE Transactions on Big Data), academic books and international conferences.

Chengjie Gu received his Ph.D. degree in Nanjing University of Posts and Telecommunications in 2012. From 2012 to 2017, he was an innovation team leader in the 38th Research Institute of CETC and conducted research and development in the communication and networking sector. Currently, he is a president of security research institute in new H3C group. He is also studying for postdoctoral fellowship at the USTC. He is a high-level innovation leader of Anhui province and a cybersecurity expert of Zhejiang province in China. His research interest includes network security and trusted network architecture, etc.

Debiao He received his Ph.D. degree in applied mathematics from School of Mathematics and Statistics, Wuhan University, Wuhan, China in 2009. He is currently a professor of the School of Cyber Science and Engineering, Wuhan University, Wuhan, China and the Shanghai Key Laboratory of Privacy Preserving Computation, MatrixElements Technologies, Shanghai 201204, China. His main research interests include cryptography and information security, in particular, cryptographic protocols. He has published over 100 research papers in refereed international journals and conferences, such as IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Information Forensics and Security, and Usenix Security Symposium. He is the recipient of the 2018 IEEE Systems Journal Best Paper Award and the 2019 IET Information Security Best Paper Award. His work has been cited more than 10000 times at Google Scholar. He is in the Editorial Board of several international journals, such as Journal of Information Security and Applications, Frontiers of Computer Science, and Human-centric Computing & Information Sciences.