LRB: Locally Repairable Blockchain for IoT Integration

Zihan Jiang, Qi Chen[ID], Zhihong Deng[ID], and He Zhang

Abstract—Recently, blockchain has become a crucial technology for addressing security concerns in the Internet of Things (IoT). However, the substantial storage requirements of blockchain present a major obstacle to integrating IoT with blockchain. This paper introduces a novel coded blockchain architecture called locally repairable blockchain (LRB) to reduce the storage costs for IoT devices. Our architecture employs a node selection allocation algorithm to determine encoding parameters and local repair groups based on the blockchain's current state, enhancing system availability. We also present a new coding process, GELRC, which combines group coding exchange methods with locally repairable codes, significantly reducing encoding complexity. GELRC facilitates low-cost local repair and improves fault tolerance. Furthermore, we introduce a specialized Vandermonde matrix for designing the local codes of LRB, enhancing the scalability of existing coded blockchains. Experimental results demonstrate that our architecture outperforms previous coded blockchain solutions with greater fault tolerance, improved single-point repair capabilities, lower coding complexity, and particularly, eliminates the need for re-encoding when new nodes are added.

Index Terms—Blockchain, storage optimization, locally repairable code, node selection and allocation, group coding exchange.

I. INTRODUCTION

INTERNET of Things (IoT) has a wide range of application scenarios, such as smart home, intelligent transportation, intelligent manufacturing, intelligent medical, and others [1]. Driven by maturing industrial use cases, the number of IoT connections is expected to reach 83 billion by 2024 [2]. Therefore, how to resist the problem of data concentration caused by centralized management and how to ensure data traceability and data security have become rigorous challenges in the areas of IoT [3], [4], [5]. Blockchain [6], with its

inherent features of decentralization, security, and integration [7], has emerged as a critical technology for addressing IoT security concerns. Nevertheless, Akrasi-Mensah et al. [8] pointed out that blockchain can solve many security problems in IoT scenarios, but it also bring huge storage consumption. In precise, typical blockchains usually use full replication data storage, namely, every full node in the network needs to store the entire blockchain ledger, thus the storage capacity of the nodes in the blockchain network will eventually be exceeded since the ledger is append-only [9], [10]. This makes it challenging for resource-constrained IoT devices to function as full nodes in the blockchain, thereby weakening the decentralized nature.

One of main techniques to solve the problems of excessive demand for storage of devices as nodes in the integration of IoT and blockchain is coded blockchain [11], [12]. This technique can reduce the storage requirements of blockchain nodes while ensuring data security and integrity [12], [13], and it can also improve block propagation efficiency [14], [15], [16]. Coded blockchain use erasure codes such as Reed-Solomon (RS) codes [17] or low-density parity-check (LDPC) codes [18] to encode the original data blocks of the blockchain into coded block. These coded blocks are then distributed and stored across multiple nodes in a decentralized manner. This approach allows each node to store only a portion of the coded blocks, significantly reducing the storage requirements for individual nodes. For example, in [11], a (n,k)-RS code is used to encode a set of k original data blocks on the permissioned blockchain into n coded blocks. Then, each node in the coded blockchain retains only the required coded blocks and discards the rest. As a result, in the coded blockchain, each node's storage requirement is reduced to approximately $\frac{1}{k}$ compared to traditional blockchain. At the same time, erasure codes possess the property of ensuring the recoverability of the original data. When a node requires a specific data block, it can collect slightly more than k coded blocks from other nodes and decode them to reconstruct the original data. Therefore, the complete blockchain data is always recoverable, guaranteeing the integrity of the blockchain data.

Although coded blockchain is an excellent technology for solving storage needs, there are still some problems:

- *High single-point repair cost.* Known coded blockchains techniques did not fix single-point errors with fewer resources. In the context of IoT, when a device failure occurs as a single-point error, a complete decoding process is required for repair. This decoding process requires interaction with a significant number of devices,
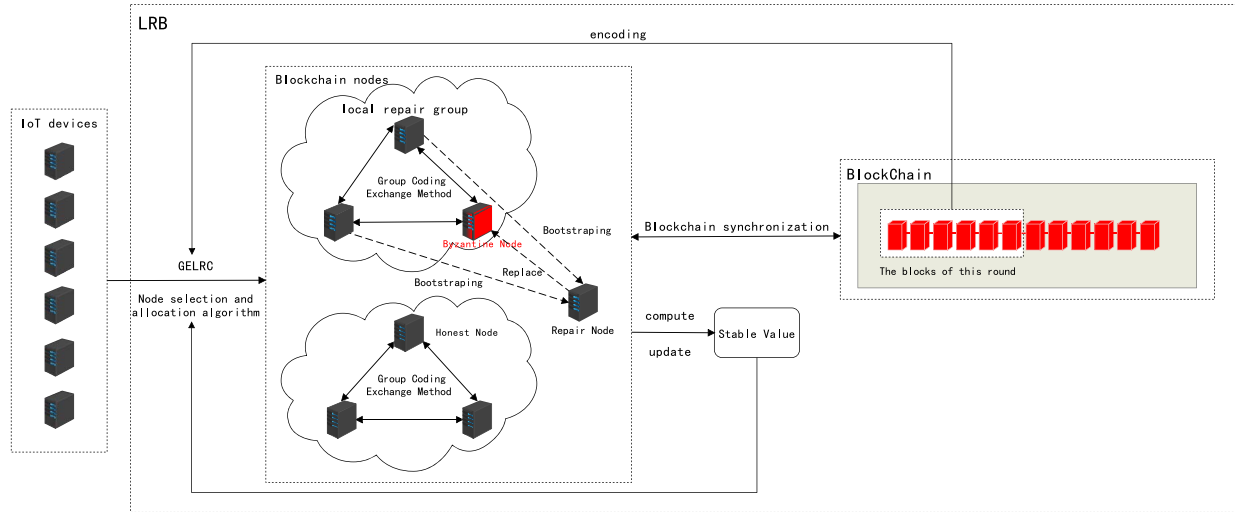
Fig. 1. The architecture of LRB.

and it has high requirements for computational resources. As a result, single-point repairs demand substantial computational and communication resources.

- *Poor scalability.* Known coded blockchain approaches using traditional erasure coding methods struggle to handle the addition of new nodes effectively. In the context of IoT, where devices undergo frequent changes, the traditional approach requires triggering a complete recoding process across all nodes whenever a new IoT device joins as a blockchain node. This places a significant burden on the performance of the blockchain system.
- *Poor availability.* Known coded blockchains techniques lacked availability. In [19], [20], [21], [22], they preconfigure coding parameters and cannot adjust them based on the specific state of the blockchain network at each time period. The fixed coding parameters approach is not suitable for the dynamic nature of device quantity in the IoT environment.

In conclusion, while coded blockchain technology effectively reduces storage requirements for individual blockchain nodes, allowing resource-constrained IoT devices to participate in blockchain consensus, it still falls short in addressing the challenges posed by the dynamic nature of IoT environments. This includes fluctuations in the number of nodes and the frequent occurrence of single-point failures. A more efficient approach is needed to minimize communication demands and conserve computational resources in such dynamic IoT settings.

To address these challenges, we propose a novel coded blockchain architecture called locally repairable Blockchain (LRB) for integrating IoT with permissioned blockchains. The key difference from known coded blockchains is that we introduce locally repairable codes (LRCs) [23], [24], [25]. A code over a finite alphabet is called LRCs if every symbol in the encoding can be recovered by a small number of other symbols of the codeword. The specific architecture of LRB is shown in Fig. 1.

In our architecture, we utilize a node selection allocation algorithm to select the parameters for each encoding and the local repair group based on the current states of the blockchain, including the current number of nodes, their stability, and the number of blocks to be encoded during the epoch. This algorithm enhances the availability of the system and guarantees that the coded blockchain satisfies the Byzantine fault-tolerant model. Moreover, we propose the GELRC coding process by combing group coding exchange method with LRCs. During the GELRC encoding process, each node performs RS pre-coding on the original data blocks generated in the current time period based on the encoding parameters and grouping situation obtained through the node selection and allocation algorithm. Then, the nodes within the local group exchange and perform XOR operations on the coded blocks to obtain the final storage content. This approach significantly reduces the computational complexity and storage requirements for individual nodes during the encoding process, ensuring resource-constrained IoT devices can become blockchain nodes.

GELRC adopts RS codes for pre-encoding, so that it has the property of Maximum Distance Separable (MDS) [26]: After collecting the required data, the global content can be restored. It also trades a small storage cost for the local repair capability, which can recover the error content within the local repair group if a small number of errors occur in the group. On the other hand, under certain conditions, GELRC can use the combination of global and local repair to provide greater fault tolerance than LT and RS. In addition, the RS pre-coding we employ is a systematic code [27], which means that the coded blocks consist of both original data blocks and encoding parity blocks. Upon decoding, the original data can be directly obtained.

To ensure the scalability of LRB, we also propose a new special Vandermonde matrix to design the local codes of LRB. This enables a new node joining scheme without the need for re-encoding by all nodes.

In summary, the main contributions of this paper is the proposal of a new coded blockchain architecture, LRB, specifically designed for the integration of permissioned blockchain with IoT. It greatly reduces the storage requirements of a single node on the premise of ensuring that blockchain data is globally recoverable. Additionally, LRB is capable of adapting to the dynamic nature of IoT environments, providing local repair capabilities with lower bandwidth and computation costs, and offering an efficient solution for incorporating new nodes. The specific contributions are as follows:

- We present a new algorithm used to the node selection and allocation. Compared with the methods in [19], [28], our algorithm determines appropriate encoding parameters and local repair groups for each epoch based on the number of blocks and nodes, and the stability of the nodes. This ensures that the encoding parameters adapt to blockchain's state.
- We propose a coding method called GELRC, which exhibits lower coding complexity and higher fault tolerance while ensuring local repair capabilities. We combine the group coding exchange method with the traditional architecture of LRCs. By using the node selection and allocation algorithm, we preassign local repair groups to specify the specific blocks for each node to encode. Nodes then obtain the final storage through intra-group exchanges and XOR operations. As a result, GELRC reduces the need for multiple RS encoding operations traditionally required in LRCs coding process to just one. At the same time, GELRC can achieve low-cost local repair and higher fault tolerance.
- We present a method of adding new nodes for blockchain. Compared with the method in [11], our method does not need to re-encode all nodes after new nodes are added, and ensures that the contents stored in the new nodes still meet the nature of global repair. The key of our method is that we propose a new special Vandermonde matrix to design the local code of LRB.

The remainder of this paper is organized as follows. Section II introduces the work related to IoT and coded blockchain. Section III introduces the background and motivation of our work. Section IV introduces the model of our work, research objectives, and explanations of the concept symbols in the paper. Section V presents the specific design of LRB, including the overall framework of the system, the working process and key functions of LRB. Section VI presents the evaluation of the specific performance of LRB. Section VII gives the comparison with previous coded blockchain. Section VIII concludes the paper.

II. RELATED WORK

Qi et al. [11] proposed the BFT-Store for permissioned blockchain system, which encodes the specified k original blocks with (n,k)-RS according to the number of nodes n in each period, and then uses multiple replication technology to save the coded blocks in the form of a full copy to each node, so as to improve the read rate. When a new node is added, in order to adapt to the adjustment of RS coding parameters, the master node collects the contents stored by each node to achieve global decoding, and then it sends the new coding parameters to all current nodes for a new round of re-encoding to ensure the participation of all nodes. In this encoding method, each node does not need to store the complete ledger, the node s just need to store individual coding blocks, which effectively reduces the storage requirement and can still recover the complete ledger through RS decoding with MDS properties. However, in an environment where nodes change frequently, the new node addition method proposed by BFT-Store requires repeated decoding and re-encoding operations, which will undoubtedly cause a huge burden on the entire blockchain system.

Regarding the problem of high storage overhead and low reliability in BFT-Store's adoption of replica storage, Zhang et al. [29] proposed a storage model based on group-based block storage. This method reduces the overall storage overhead in blockchain systems by using block splitting and unit encoding. By dividing a single block into multiple fragments, encoding them and storing them on nodes across multiple shards [30], the original data and the parity blocks are saved on multiple shards. This not only saves storage space but also greatly enhances data reliability, as any shard storing an coded block can recover the original data through decoding. However, this method also fails to consider the scenario of node join and exit. The changes in nodes within the shards result in frequent encoding and decoding operations, which impose a significant burden on the blockchain system. This imposes a significant burden on the blockchain system. Additionally, the security of sharding is difficult to guarantee. The security level of the sharding approach is directly related to the group size [31], which will compromise the security and integrity of the blockchain to some extent.

Kadhe et al. [19] proposed SeF, a blockchain architecture that uses LT code [32]. It performs LT code for the specified k blocks in a certain period of time, and each node generates s coded blocks independently according to the degree distribution of LT code. When $s = 1$, the maximum storage saving is achieved, that is, each node only needs to store $\frac{1}{k}$ original content. Because of the particularity of the LT code, when a new node joins, it only needs to select a degree according to the degree distribution and complete the encoding independently, without re-encoding of all nodes. However, LT decoding needs to contact more nodes and cannot guarantee the success of decoding. Therefore, Tiwari and Lalitha [28] propose the S-RED architecture based on SeF, which adopts the Raptor codes of the two-layer architecture. It combines an erasure codes and LT code. It uses LDPC codes as pre-encoding first to make subsequent LT code adopt a simpler degree distribution to reduce the decoding cost and improve decoding success rate by using two-step decoding.

While the aforementioned approaches based on RS codes, LT codes and Raptor codes effectively achieve storage reduction, they share a common challenge when it comes to potential sporadic single-point failures in the blockchain network. Repairing the content of a damaged node requires a complete decoding process, which incurs a significant cost.

Recently, coded blockchain has been applied to IoT. Singh et al. [20] propose a medical blockchain that uses a combination of LT code and dynamic region partitioning [33] to reduce the storage requirements of a single IoT device and enable decoding with lower communication cost. However, this scheme does not work well in a dynamic blockchain environment, and it is difficult to cope with changes in the number of nodes.

III. BACKGROUND AND MOTIVATION

We first introduce the integration of IoT and blockchain, along with the development of coded blockchain. Then, we present the motivation of our work.

A. Integration of IoT and Blockchain

IoT links physical devices, sensors, software, and other components to the Internet, enabling communication and data exchange between devices. It provides functionalities such as real-time monitoring, remote control, automation, and intelligent operations. With its broad range of applications, IoT has seen rapid growth in recent years, leading to a surge in IoT connections and the generation of massive amounts of data.

Blockchain technology offers a promising solution to the challenges faced by IoT. Its heightened security, increased transparency, and traceability make it an ideal fit for integration with IoT, which has long struggled with security concerns. In particular, permissioned blockchains, though less decentralized than permissionless blockchains, offer an access control design that is well-suited for managing device admission in the IoT [34]. Furthermore, unlike permissionless blockchains with low throughput that fall short of IoT requirements, permissioned blockchains can achieve higher throughput by allowing only validated nodes to join and utilizing more efficient consensus mechanisms [35]. As a result, permissioned blockchains are better suited for integration with IoT. Their superior features have made them the main solution in recent years for integrating blockchain technology with IoT [36], [37]. For example, Feng et al. [38] combined IoT with permissioned blockchain to propose a blockchain-based intelligent 5G drone Internet cross-domain authentication system. By combining consortium blockchain [39], they put forward a secure and effective distributed authentication mechanism suitable for IoT. This ensures mutual authentication between IoT devices in the distributed environment.

Although integrating IoT with blockchain is a promising solution, it faces a significant challenge: the large size of blockchain ledger data. This issue requires full blockchain nodes to store vast amounts of data, which is difficult for resource-constrained IoT devices to manage.

B. Coded Blockchain

To preserve its decentralization, anti-tampering, and traceability features, blockchain must continually manage its expanding data. As a result, nodes require increasingly larger storage resources to maintain blockchain data. Nodes with limited storage capacity may be forced to exit the blockchain

system, undermining its decentralization and limiting the application and development of blockchain technology [40]. This challenge is especially significant in the integration of IoT and blockchain, where resource-constrained IoT devices acting as blockchain nodes struggle with data storage demands.

Coded blockchain architecture offers an effective solution to the challenges mentioned earlier. Drawing from research in distributed storage [41], it utilizes erasure codes to encode blocks, which are then stored in a distributed manner. This approach allows each node to store only coded fragments, significantly reducing both data storage requirements and communication costs [42]. The repairable nature of erasure codes ensures that original block data can be recovered by collecting a certain amount of data, thereby maintaining the integrity of the blockchain. For instance, Yang et al. [43] integrated coded blockchain with IoT, using rateless raptor codes [44] to encode and distribute the original blockchain data across nodes. This method significantly reduces the storage demands on individual IoT devices.

C. Motivation

As previously mentioned, one of main approaches to addressing the high storage requirements for individual nodes is the use of coded blockchain architecture. This technology effectively enables resource-constrained IoT devices to function as blockchain nodes. However, while erasure coding improves storage efficiency, it also comes with the drawback of high repair costs. Repairing or replacing a single faulty node requires retrieving data from multiple available nodes for reconstruction, significantly increasing the demand for bandwidth resources [45]. Single-point failures are common in the IoT context [46], leading to frequent data reconstructions by IoT devices acting as blockchain nodes. This process demands substantial computational and communication resources, placing an excessive burden on resource-limited IoT devices. Additionally, current coded blockchain systems have predefined coding parameters and struggle to accommodate the addition of new nodes, making them less suitable for the dynamic and evolving nature of IoT environments [43].

Therefore, the challenges discussed above have inspired us to design a coded blockchain architecture called LRB, specifically tailored for the integration of permissioned blockchain with IoT. LRB is designed to reduce storage pressure on each node while enhancing storage efficiency. Additionally, LRB aims to adapt to the dynamic IoT environment through parameter adjustment strategies, provide local repair capabilities with lower bandwidth and computational costs, and offer a more efficient solution for integrating new nodes.

IV. PROBLEM SETTINGS

In this work, we focus on reducing the storage costs associated with blockchain historical data and the repair costs of damage to individual nodes. Our goal is to design a protocol that enables a full node to save its storage space and still be able to help guide a new node to recover the full data. At the same time, it also has the capability of local repair, and can repair a specific damaged node with data stored in

a small number of nodes, when certain conditions are met. Secondly, we hope that the encoding process meets high fault tolerance and low computational complexity while ensuring the scalability of the blockchain.

A. Model

Since our work is based on permissioned blockchains and grouping mechanisms, we make similar model assumptions after referring to relevant works [47], [48], especially BFT-Store [11]. We also use the PBFT consensus, which is one of the most commonly used BFT protocols in permissioned blockchains. Furthermore, Brotsis et al. [35] has confirmed that high-performance BFT protocols are suitable for IoT environments.

- *Types of participants.* The nodes in the system are either honest nodes or Byzantine nodes. Honest nodes always comply with the specified protocol and deliver the correct messages to other honest participants within a known time $\triangle$. Byzantine nodes, on the other hand, are controlled by an adversary. These malicious nodes may collude with each other and deviate from the protocol in arbitrary ways. According to the requirements of PBFT, we assume that at any given time, the number of Byzantine nodes does not exceed one-third of the total number of nodes. In other words, when there are n nodes, at most $f = \lfloor \frac{n-1}{3} \rfloor$ nodes can be controlled by the Byzantine adversary [49].
- *Network.* Our work is based on a permissioned blockchain where participants need to be verified when joining and leaving the system, which to some extent ensures the trustworthiness of participants. Additionally, our system adopts a partial synchronization model [50], utilizing a known bound $\triangle$ and an unknown global stable time. After the global stable time, all transmissions between two honest nodes are guaranteed to arrive within $\triangle$ time.
- *Delayed adaptive adversary.* We assume an adversary that is a delayed adaptive adversary [47], [48], meaning that the adversary requires a significant amount of time to corrupt a node. This assumption is adopted by most committee-based and group-based designs [51], [52]. Specifically, we assume that even if the adversary starts corrupting members as soon as they appear, by the time they can successfully corrupt a member, the current round of encoding has already ended or the encoding process has moved on to the next round, and the member is no longer in the same group.

B. Goals

While previous encoding-based approaches have improved storage efficiency, they also impose significant bandwidth requirements. Especially when there is a single-point error, the node needs to obtain a large amount of data from other nodes enough to decode the entire ledger, which leads to a decrease in the performance of the blockchain. At the same time, they also lack a solution that can deal with the problem of requiring all nodes to re-encode after new nodes join the network.

So our LRB has the following goals:

- *Scalability.* After new nodes join and decode to recover the complete blockchain, they can independently perform encoding without requiring other nodes to undergo re-encoding, thereby reducing the burden on the blockchain network.
- *Fault tolerance.* In the face of attacks by Byzantine opponents, that is, when malicious nodes collude with each other, it can still ensure the recovery of blockchain data, and provide greater fault tolerance than previous work using encoding.
- *Availability.* It is possible for our LRB to choose better encoding parameters and encoding block distribution strategies based on the specific state of the blockchain network to ensure that LRB satisfies Byzantine fault tolerance, while keeping the encoding complexity low.
- *Local repair capability.* Under the premise of ensuring the availability of block data and the scalability of blockchain nodes, the cost of single-point repair is reduced compared with traditional coding methods.

C. Concepts and Notations

In blockchain, we will choose coding parameters based on the number of nodes in the blockchain network. Here, we set n coded blocks to be distributed and stored in n blockchain nodes, with each node storing one block. Therefore, in the following concepts and notations explanations, n represents both the number of coded blocks and the number of nodes.

- MDS is short for Maximun Distance Separable. A (k, n-k)-MDS code with code rate $R = \frac{k}{n}$ divides a file of size M into k blocks of the same size, and then encodes them into n blocks of size $\frac{M}{k}$. MDS code ensures that any k of these n blocks can be used to reconstruct the complete original data.
- (n, r, d, M, a)-LRCs is a coding method with parameters. In the blockchain, we can understand n as the number of nodes, r as the local coding parameter, d as the coding distance, and M as the number of original data blocks, a is the size of each encoding result. The LRCs can also achieve global repair [23], [24], [25], and its global repair parameter $k = \frac{M}{r}$, so we subsequently simplify this encoding to (n, r, k, M)-LRCs. Since GELRC shares the same parameter selection as LRCs, we will not elaborate on it further.
- (n, k)-RS refers to the Reed-Solomon code with parameters n and k. In the blockchain, we can understand n as the number of nodes and k as the number of original data blocks. Through RS codes, k original data blocks will be encoded into n coded blocks, including k original blocks and $n-k$ check blocks. RS codes have the property of MDS, that is, only any k blocks among n coded blocks are needed to recover the complete original data.

V. LRB ARCHITECTURE

This section mainly introduces the specific design of LRB. We start from the general architecture of LRCs we use to introduce the specific coding process of LRCs. Next, we introduce

the difference between GELRC and conventional LRCs: the group coding exchange method. For decoding, we proposed solutions for two different scenarios. We also introduced the encoding scheme used when new nodes join. Finally, we explained the node selection and allocation algorithm adopted in this paper.

A. Generic Framework of LRCs

Papailiopoulos and Dimakis [23] propose (n, r, d, M, a)-LRCs. For detailed parameter explanations, please refer to Section IV-C. LRCs can store M size files into n nodes by encoding, and realize locally recovery capability of $r + 1$ nodes as a group and global data repair capability through the content stored by k nodes. The specific coding scheme is as follows:

- Step1: According to the local parameter r of the LRCs, the file x of size $M = r \times k$ is divided into r parts, each of which is k in size.
- Step2: Use the MDS code with parameters (n, k) to encode each part independently to obtain a set of $y^{(i)}$

$$y^{(1)} = x^{(1)}G, \ldots, y^{(r)} = x^{(r)}G, \tag{1}$$

where G is an $n \times k$ MDS generator matrix.

- Step3: We then generate a single parity XOR vector from all the coded vectors:

$$\mathbf{s} = \bigoplus_{i=1}^{r} \mathbf{y}^{(i)} \tag{2}$$

- Step4: The number of coded blocks obtained in the second step is $r \times n$, and the number of coded blocks obtained in the third step is r, so we distribute a total of $(r + 1)n$ coded blocks to n nodes for storage, and each node will get $r + 1$ blocks. The allocation of these blocks has 3 specific requirements as follows:
  - Each node contains r coded blocks coming from different $y^{(l)}$ coded vectors and 1 additional parity symbol.
  - The blocks in the $r + 1$ nodes of the i-th $(r + 1)$-group have indices that appear only in that specific repair group.
  - The blocks of each row have indices that obey a circular pattern, i.e., the first row of symbols has index ordering $\{1, 2, \ldots, r + 1\}$, the second has ordering $\{2, 3, \ldots, r + 1, 1\}$, and so on.

B. Design of LRB

We first assume a discrete time system, where time is divided into discrete time periods $T = \{1, 2, \ldots, t, \ldots, T\}$, each time period t has a fixed duration. The purpose of LRB is to use LRCs to encode l blocks generated in each time period t and store them in n nodes, so that the content stored in each node has the ability of local recovery within the group and global recovery ability under MDS conditions. At the same time, in LRB, the GELRC adopts group coding exchange method to reduce the computing resources required for coding. Moreover, LRB can realize single-point repair while ensuring
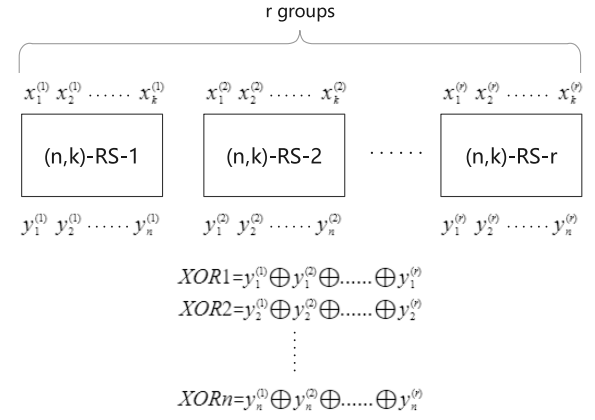

Fig. 2. Generic framework of LRCs.

that newly added nodes can complete coding independently and also have global repair capability.

1) Encoding: Each time period t corresponds to L generated blocks and N nodes in the current network. L and N may be different in each specific time period. Since our method is applicable to each specific time period, we only consider a single time period t as an example.

a) Encoding parameter determination: We assume that the number of blocks generated in time period t is L, and the number of nodes at this time is N. According to the coding requirements of LRCs, we need to divide L into r parts, each part has k blocks, and set $\frac{N}{r+1}$ local repair groups, so there are two coding conditions $r| L$, $(r + 1)| N$. In order to meet these requirements and ensure that LRB satisfies the Byzantine fault-tolerant model, we need to choose the encoding parameters. We use a node selection and allocation algorithm to obtain the final encoding parameters: the number of original data blocks l of this period of encoding, n nodes participating in this period of encoding and $\frac{n}{r+1}$ local repair groups, where each node stores $r + 1$ blocks, and the number of nodes in each group is $r + 1$. The detailed process of node selection and allocation algorithm is introduced in the fourth part of this subsection.

b) GELRC with group coding exchange method: If we directly apply the general framework of LRCs to the blockchain, as shown in Fig. 2. Obviously, a node needs to go through two steps of MDS encoding and XOR calculation from the beginning of encoding to the final stored result. Because of grouping, the whole process needs to calculate r MDS coding and one XOR, and the computing resource consumption of this coding method is intolerable. Therefore, we propose GELRC with the utilization of the group coding exchange method. It aims to allocate nodes in each group to r different parts for MDS encoding. Here we choose to use RS code. During the encoding process, each node only needs to perform RS encoding once and save the r coded blocks required by its respective group, and then through the way of intra-group transfer, each node obtains its own required blocks and completes XOR calculation and storage.

Next, we introduce the implementation of the GELRC in detail:

First, according to the node selection and allocation algorithm, we obtain $\frac{n}{r+1}$ local repair groups:
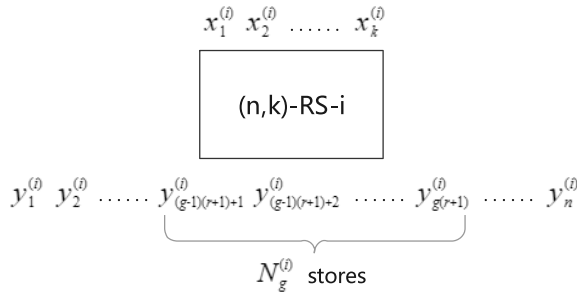
Fig. 3. RS encoding process for $R(i)$ segments.

$$G = \{G(1), G(2), \ldots, G(n/r + 1)\}. \tag{3}$$

There are $r + 1$ nodes in each group, so we define the nodes in each local repair group as:

$$N_g = \left\{ N_g^{(1)}, \ N_g^{(2)}, \ldots, \ N_g^{(r+1)} \right\}, \tag{4}$$

$g \in [1, \frac{n}{r+1}]$ represents the partial group number.

At the same time, during the GELRC encoding process, we select l blocks in sequential order from the uncoded blocks based on the result obtained from the node selection and allocation algorithm for this period. We designate these l selected blocks as the original data blocks for this period. Furthermore, we divide the l original blocks into r parts, with each part consisting of k blocks. These parts are defined as:

$$R = \{R(1), R(2), \ldots, R(r)\}. \tag{5}$$

Then we map N to R:

$$y\left(N_g^{(i)}\right) = R(i), i \in [1, r]. \tag{6}$$

This mapping indicates that the first r nodes in each group correspond to r parts into which the original block is divided.

In this method, a node only needs to perform RS coding once in each round of coding. We take a general node in the coding process as an example. Assuming that the node is the i-th node belonging to the g-th group, denoted as $N_g^{(i)}$. According to the mapping between N and R in formula (6), we get the input block group $R(i)$ corresponding to this node, and the structure of $R(i)$ is shown in the upper side of Fig. 3. Fig. 3 shows the RS encoding process of the $R(i)$ segment. The upper side is the input k original blocks. Through (n,k)-RS encoding, we obtain the coded block group $Y = \{y_1^{(i)}, y_2^{(i)}, \ldots, y_n^{(i)}\}$. The encoding process of our node $N_g^{(i)}$ is to save some coded blocks $Y(N_g^{(i)})$ required by this group after the input of the original blocks of $R(i)$ is encoded by RS code, where $Y(N_g^{(i)})$ is denoted as:

$$Y\left(N_g^{(i)}\right) = \left\{ y_{(g-1)(r+1)+1}^{(i)}, y_{(g-1)(r+1)+2}^{(i)}, \ldots, y_{g(r+1)}^{(i)} \right\}. \tag{7}$$

Once the r nodes in each local repair group complete their encoding process, the final block distribution can be achieved by performing data exchange within the group and executing an XOR operation on each node.

Example: The parameters of one period of encoding are $l=12$, $n=9$, $r=2$, we obtain the original block grouping $R(1)$

and $R(2)$ and three local repair groups, each local repair group has 3 nodes. Fig. 4(a) shows the encoding process, and Fig. 4(b) illustrates the process where nodes achieve the final storage state through intra-group exchanges and XOR operations. The general framework of LRCs requires each node to perform two (9, 6)-RS encoding operations under the same parameters in the given examples. In contrast, our proposed group coding exchange method only requires each node to perform a single RS encoding operation regardless of the encoding parameters.

As a matter of fact, the encoding complexity of our method is significantly lower than that of the traditional LRCs framework. More details and comparative experiments can be found in Section VII-C.

2) Decoding: The decoding in LRB is divided into three cases: original data reconstruction, local restoration decoding, and local and global cooperative decoding.

a) Original data reconstruction: Because each part of a period is encoded in (n, k)-RS, when a node attempts to recover all the original data blocks in a period, it needs to contact at least k nodes, of which any node in $\frac{n}{r+1}$ groups can satisfy the requirement. These k nodes can recover the original data block through the decoding process of RS code. As long as there is a bootstrap cost that satisfies decoding needs, LRB can guarantee successful recovery of the original data.

b) Local restoration decoding: It is well known that coded blockchains utilizing several commonly used MDS coding schemes lack the capability for local repair. Although the use of coding reduces the storage of individual nodes, when there is a need to recover individual data loss, you must use global decoding, which greatly increases the demand for bandwidth and computing resources. Therefore, in order to have better ability to recover the original data, some systems still choose to save backup copies of the encoded results. However, due to the adoption of GELRC for encoding, LRB exhibits enhanced local repair capabilities. Each $r + 1$ node in the LRB group has a local repair capability with parameter r. Therefore, when a new node is added to replace the abnormal node, only r nodes in the group need to be contacted for local repair, and the local repair process only involves the operation of coding block XOR, so the computational complexity is linear.

c) Locally and globally cooperative decoding: Conventional (n, k)-MDS coding can tolerate a maximum of $n-k$ data block failures, beyond which the data becomes permanently irrecoverable. However, with LRB, under certain distribution conditions of the damaged blocks, it can achieve a higher fault tolerance, tolerating more than $n-k$ failures. Assuming that the first $n-k$ nodes in LRB are corrupted, and each remaining local encoding group has one node damaged, this scenario represents the maximum fault tolerance of LRB, with the maximum number of node failures being:

$$n - k + \left\lfloor \frac{k}{r+1} \right\rfloor. \tag{8}$$

3) Scheme for New Node Joining Without Re-Encoding: Since the number of nodes in the LRB is dynamically changing, we need to consider how new nodes are encoded
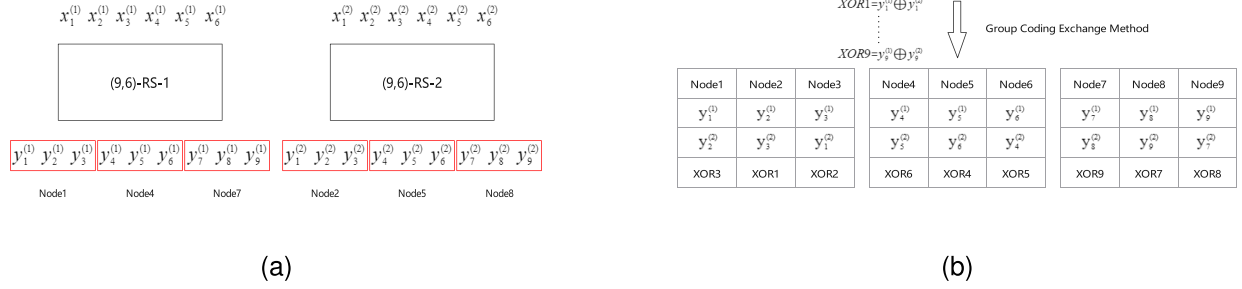
Fig. 4. Group coding exchange method. (a) The encoding process. (b) Intra-group exchanges and XOR operations.
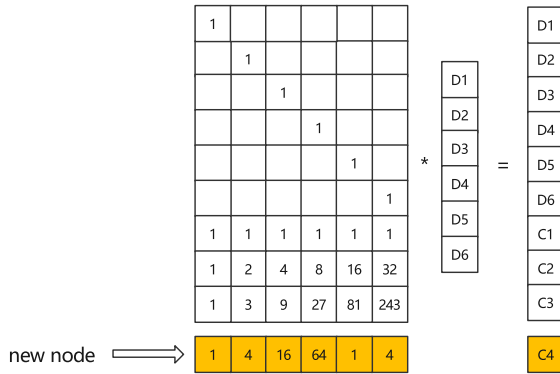
Fig. 5. Independent encoding process of the new node in a single original block group.

after they join. In essence, there are two cases of adding a new node: replacing a damaged node and adding a new node. The former can be realized through local restoration decoding and will not be repeated here. We describe the latter process in detail.

The RS re-encoding adopted in [11] requires the participation of all nodes, which will lead to very high consumption, so we adopt a new node joining scheme that does not require all nodes to recode. The Vandermonde matrix used in the RS pre-encoding process of LRB needs to be specially designed. We specify the rows in the Vandermonde matrix that generate redundant blocks. As shown in Fig. 5, we select the rows of the coding matrix in an increasing order starting from 1. In this example, 3 coded blocks are generated, so the rows of the coding matrix are composed of power sets of 1, 2, and 3 respectively. The coding matrix is the same for all original block groups. When a new node is added, we need an additional block, so we still add a row to the encoding matrix in increasing order (the row added in the example is a power set of 4) and calculate the new redundant block with this row. After the new node has computed the new blocks for all the original block groups, it performs one XOR operation on these blocks and saves the XOR result with the coded blocks. Therefore, when a new node joins, it only needs to perform independent encoding, and only the coding matrix is updated for other nodes.

Newly added nodes encoded in this way have the same global repair capability as other nodes, but before the number of newly added nodes meets $r+1$ in the current period and completes data exchange, these nodes have no locally repairable ability.

4) Node Selection and Allocation Algorithm: This part primarily introduces the operational process of the node selection and allocation algorithm. This algorithm determines the local repair parameter r, based on the number of blockchain nodes L, the stability of these nodes, and the number of blocks to be encoded N, during a given time period. The parameter r is chosen to satisfy $r|L$ and $r+1|N$, while ensuring that our GELRC can achieve the recovery of global data even when there are a maximum of $f = \lfloor \frac{n-1}{3} \rfloor$ malicious nodes.

Similar to traditional LRCs, the global repair capability of GELRC is primarily provided by (n, k)-RS pre-encoding. Since the original L blocks are divided into r parts for encoding, the parameter k is defined as $k = \frac{L}{r}$. Therefore, when selecting the local parameter r, we only need to ensure that the error-correcting capability of the encoding satisfies $n - \frac{L}{r} \geq f$, where f represents the maximum number of Byzantine faults. By meeting this condition, the Byzantine fault-tolerant model is satisfied.

In order to better distinguish node states, the algorithm uses stable value (STBV), which is a constant between 0 and 1, to represent the stability of each node. The higher the stable value, the more likely the current node is to respond to the request and provide the correct storage information. We will take the time from the end of the last consensus round to the end of the current consensus as a period, and the stable value will be updated once in each period. We are given a general representation of STBV:

$$\text{STBV}_i^t = \text{STBV}_i^{t-1} + x \times \text{NORMAL} + y \times \text{ERROR}, \quad (9)$$

where STBV_i^t represents the stable value obtained by the current node i at the end of the current set period t. It is determined by the stability of the previous period, the number of normal responses and correct content returned in this period, and the number of non-responses or incorrect content returned in this period, respectively, represented by STBV_i^{t-1}, NORMAL, ERROR. The x and y in the formula (9) represent rewards and punishments, respectively, and they

Algorithm 1 Determine the Number of Blocks to be Encoded in This Period

---

Input: The sum L of the number of blocks in this period and the number of remaining blocks in the previous period.

Output: The number of original data blocks encoded in this period l.

1: **if** L is a prime number **then**
2:      **repeat**
3:          // Subtract blocks from the back until a non-prime number is found.
4:          $L \leftarrow L - 1$
5:      **until** L is not a prime number
6: **end if**
7: $l \leftarrow L$
8: **return** l

---

are both small real numbers that can be adjusted according to demand. In order to increase the cost of evil actions and to identify malicious/corrupt nodes more quickly, we usually require $y > x$. At the same time, we need to deal with malicious/damaged nodes, so when the STBV of a node is lower than 0.2, we consider the node to be an extremely unstable node or a malicious node, and it will be replaced the next time a new node is added. Such a replacement process can be treated as dealing with a simple single-point error, and local restoration decoding can be used to achieve inexpensive recovery of stored contents.

The node selection and allocation algorithm consists of three steps. The input parameters are: the sum L of the number of blocks in this period and the number of remaining blocks in the previous period, all nodes N and their STBV.

First, based on the conditions $r|L$ and $(r + 1)|N$, we ensure that the number of original data blocks is not a prime number. If it is a prime number, we reduce the number of blocks and transfer these blocks to the next time period for encoding. The algorithm to get the number of original data blocks encoded in this period l is in Alg. 1.

Next, based on the conditions $r|l$ and $(r + 1)|N$, we adjust the number of participating nodes for encoding. We sequentially reduce the nodes based on the current N until the condition is met. At this point, we obtain the parameters n, r, and k required for this round of encoding. Then, we calculate the maximum allowable number of error nodes for n nodes based on $f = \lfloor \frac{n-1}{3} \rfloor$ and compare it with the error correction capability of the RS pre-coding, that is, whether the condition $n - k \geq f$ is satisfied. If it is satisfied, it proves that our coded blockchain can tolerate a maximum of f arbitrary faulty nodes under the condition of n nodes, which complies with the Byzantine fault tolerance model. If it is not satisfied, decrease N and continue searching for parameters. The algorithm for obtaining the encoding parameters and the number of participating nodes is shown in Alg. 2.

Based on the obtained parameters n and r, this algorithm will select the required n nodes for this period of encoding from all nodes N in the current blockchain network in descending order of STBV. The n nodes are then divided into $r + 1$ groups according to the previous sorting order.

Algorithm 2 Determine the Number of Nodes Participating in this Period of Encoding

---

Input: The number of original data blocks encoded in this period l. The number of nodes in the current blockchain network N.

Output: The number of nodes participating in the encoding n. Encode parameters r and k.

1: // Determine the encoding parameter r based on the condition $r|l$, $(r+1)|N$. If it is not possible to find an r that satisfies the condition, reduce the number of participating nodes N.
2: $r \leftarrow \max(l, N) + 1$
3: **repeat**
4:      **for** $i = 2$ to $Min(l, N)$ **do**
5:          **if** $l\%i == 0$ **AND** $N\%(i+1) == 0$ **AND** $l/i < N$ **then**
6:              $r \leftarrow i$
7:              $n \leftarrow N$
8:              $k \leftarrow \frac{l}{r}$
9:              **break**
10:          **end if**
11:      **end for**
12:      **if** $i >= \min(l, N)$ **then**
13:          $N \leftarrow N - 1$
14:      **end if**
15: **until** $r < \min(l, N)$
16: $f \leftarrow \lfloor \frac{n-1}{3} \rfloor$
17: **if** $n - k < f$ **then**
18:      Back to step 4 and let $i = r + 1$
19: **end if**

Algorithm 3 Determine the Local Repair Groups

Input: All nodes N and their STBV. The number of nodes participating in the encoding n. Encode parameters r.

Output: $\frac{n}{r+1}$ local repair groups G.

1: // Sort N nodes in descending order based on STBV and select the first n nodes. We define this node as N_e.
2: $N_e \leftarrow$ **SORT**(N, STBV, n)
3: // Divide the n nodes into $r + 1$ groups according to the order of the sorting result. We defined this groups as S
4: $S \leftarrow$ **DIVEDE**(N_e)
5: **for** $i = 1$ to $\frac{n}{r+1}$ **do**
6:      **for** $j = 1$ to $r + 1$ **do**
7:          Randomly select one node from S_j and add it to the local repair group G_i.
8:      **end for**
9: **end for**
10: **return** G

Finally, we randomly select one node from each group to form a local repair group. This process is repeated to obtain $\frac{n}{r+1}$ local repair groups. This approach ensures both randomness in grouping and dispersion of low-stability nodes across different local repair groups. For more details, please refer to Alg. 3.

VI. PERFORMANCE MEASURES

A. Storage Savings

Storage savings is defined as the ratio of the size of the full ledger (The data stored by traditional blockchain full nodes) to the size of the blockchain ledger actually stored by each node. In each epoch, for l original data blocks, each node of the LRB needs to store $r+1$ coded blocks. Although our LRB can only achieve

$$\gamma = \frac{\text{size}(B)}{\text{size}(\text{Enc}(B,j))} = \frac{l}{r+1}$$

in storage savings compared to the maximum achievable storage savings $\gamma = \frac{\text{size}(B)}{\text{size}(\text{Enc}(B,j))} = l$ in the coded blockchains using RS or LT encoding, our LRB has the ability for local repair and higher fault tolerance while saving storage.

In fact, in order to ensure security, the blockchain using RS code will store coded blocks by copying [11], so it generally cannot achieve optimal storage. On the other hand, the blockchain using the LT code needs a large number of nodes to support the maximum storage. To meet the requirement that the number of storage nodes must be greater than the number of original data blocks, it is also necessary to make a trade-off for storage savings when the number of nodes is insufficient, while our LRB does not have this requirement.

B. Local Repair Capability

This part corresponds to our goal of having local repair capability in LRB. More detailed comparative experiment can be found in Section VII-B.

We use the bootstrap cost to compare the repair capabilities of BFT-Store, SeF and LRB. The bootstrap cost is defined as the number of honest nodes that a new node needs to contact in order to restore the original data when repairing a node or bootstrapping a replacement node within a specific period. In BFT-Store, a node needs to download l undamaged coded blocks to join the blockchain network or replace a damaged node. The LT code used in SeF does not guarantee that the original data can be recovered through l coded blocks. Considering the encoding scheme with storage savings of γ in LT code, for a given probability of decoding failure $0 < \delta < 1$, if a node stores s coded blocks per turn, the recovery cost of historical blocks in the blockchain requires that the decoding nodes, which collect data for decoding, interact with a minimum of

$$K(\gamma, \delta) = \frac{l + O\left(\sqrt{l}\ln^2(\frac{l}{\delta})\right)}{s}$$

honest nodes that store coded blocks. This means that $l + O(\sqrt{l}\ln^2(\frac{l}{\delta}))$ honest nodes need to be contacted for repair at the maximum storage savings.

For LRB, we assume that the vast majority of nodes are honest, with only a small number of malicious or corrupt nodes. The local repair capability of LRB can tolerate up to $f_{\text{local}} = \frac{n}{r+1}$, which means each local repair group can damage one node. In this case, only $r(r \ll l)$ nodes in each group need to be contacted to complete the repair of a single node in the group. Since the node selection and allocation algorithm is adopted in LRB, nodes with low STBV among the whole n nodes are dispersed into each local repair group. Therefore, we believe that even if the node damage occurs, it is unlikely that the local repair ability of the local repair group will be directly affected.

Therefore, based on the aforementioned analysis, we believe that compared to BFT-Store and SeF, our LRB achieves local repair with low bandwidth and computational costs at very low bootstrap cost. When $f_{\text{local}} > \frac{n}{r+1}$, LRB can no longer use local repair to recover data. At this time, we can still restore global data through RS decoding.

C. Fault Tolerance of Decoding

This part corresponds to our goal of having fault tolerance in LRB when facing malicious nodes or node loss. The comparative experiment can be found in Section VII-A.

When a new node joins or when LRB loses its local repair capability, we need to decode the complete blockchain. As mentioned earlier, both SeF using LT encoding and BFT-Store using RS encoding lack local repair capability, so they always need to restore all data. In a period, RS and SeF with a given decoding failure probability $0 < \delta < 1$ respectively need to communicate with l and $l + O(\sqrt{l}\ln^2(\frac{l}{\delta}))$ honest nodes. However, the GELRC adopted by LRB uses multi-segment (n, k)-RS for coding, so it has a global repair capability with parameter k. Since we use the node selection and allocation algorithm to choose encoding parameters that guarantee the coded blockchain satisfies the Byzantine fault tolerance model, we can contact k ($k<l$) nodes among the $n - f(f \leq \lfloor \frac{n-1}{3} \rfloor)$ honest nodes for repair. In fact, due to the local repair capability of LRB, decoding can be accomplished with fewer than k nodes in specific cases.

In order to compare the fault tolerance of the three encoding methods, we define the fault tolerance of decoding as the maximum number of nodes that can be lost or damaged. We assume that there are n nodes and l original blocks in a period, where $n > l$. Since the LT code used by SeF needs to contact more than l nodes to recover, its decoding error tolerance rate is less than $n-l$, and (n, l)-RS code has MDS property, so its error tolerance is fixed at $n-l$. However, LRB using (n, r, k, l)-GELRC can realize $n-k$ fault tolerance when any node is lost. In the optimal case: Assuming that $n-k$ nodes are lost continuously, each of the remaining local repair groups $\lfloor \frac{k}{r+1} \rfloor$ is still allowed to lose one. In this ideal situation, the fault tolerance can reach $n - k + \lfloor \frac{k}{r+1} \rfloor$. Therefore, our LRB has a stronger fault tolerance capability compared to BFT-Store and SeF.

Proof: Suppose there are n nodes, the local parameter is r, and the global recovery parameter is k. Because GELRC uses multiple (n, k)-RS, data can be recovered if $n-k$ nodes are lost. We obtain the maximum number of complete local repair groups after $n-k$ nodes are lost: $\lfloor \frac{k}{r+1} \rfloor$. In the optimal case, one node is lost in each of these complete local repair groups, so that the lost nodes can be locally repaired by contacting the remaining r nodes in the group, so we have $\lfloor \frac{k}{r+1} \rfloor$ nodes for additional fault tolerance. Therefore, the LRB is ideally fault-tolerant up to $n - k + \lfloor \frac{k}{r+1} \rfloor$.

D. Coding Complexity

This part corresponds to our goal of achieving availability in LRB. The comparative experiment can be found in Section VII-C.

Our GELRC can select encoding parameters based on the blockchain state. It reduces the encoding complexity of the traditional LRCs by approximately a factor of r (where r is the number of groups in the encoding parameters), while still ensuring that the coded blockchain complies with the Byzantine fault tolerance model.

The traditional LRCs encoding architecture divides the original blocks into groups based on the encoding parameter r and performs RS encoding and XOR operations on each group separately. This architecture requires each node to perform r rounds of RS encoding. In contrast, our GELRC achieves a single RS encoding operation for each node, regardless of the encoding parameter r, through group encoding and intra-group exchange.

In summary, the GELRC coding process used in our LRB significantly reduces the coding complexity of individual nodes compared to traditional LRCs.

E. New Node Joining Scheme

This part primarily compares the re-encoding mechanism for adding new nodes in the traditional RS encoding scheme with our proposed LRB architecture, which allows new nodes to join without the need for re-encoding. It corresponds to the scalability goal of our proposed LRB architecture.

In BFT-Store [11] the traditional RS code is used. Whenever a new node joins, the master node performs global decoding and then notifies all nodes in the blockchain to re-encode blocks for each period using the new parameters. If new nodes are added frequently, such frequent re-encoding will cause a great burden on the entire blockchain system. Re-encoding in BFT-Store using (n, l)-RS requires performing RS encoding processes equivalent to n times.

LRB uses a special Vandermonde matrix, which is beneficial to the encoding when new nodes join. When a new node joins, it also needs to perform global decoding first, and then it only needs to obtain a new row of the encoding matrix according to the parameters of the period encoding and the current number of nodes, and then perform RS encoding and XOR operation independently. Our LRB adopts (n, r, k, l)-GELRC. When a new node joins, it performs RS encoding on r individual original block groups to compute new blocks, and then performs an XOR operation on the obtained new blocks. Because $r \ll n$, the number of RS encoding operations is significantly reduced. If the number of nodes simultaneously added reaches $r + 1$, through group coding exchange method, each new node actually only needs to perform one RS encoding and one XOR operation, achieving a lower encoding complexity. The entire encoding process of the new node does not require re-encoding by other nodes, thus it does not impose a significant burden on the blockchain system.

In summary, our proposed solution eliminates the need for all blockchain nodes to re-encode when a new node joins. This is particularly beneficial in dynamic environments with
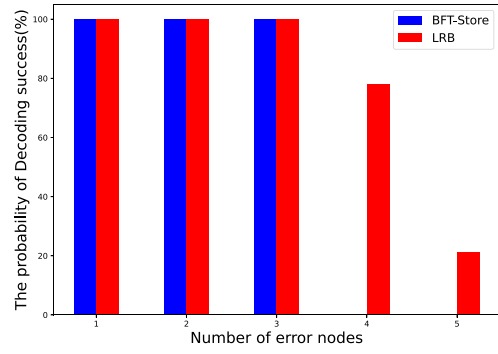

Fig. 6. Comparison of fault tolerance between GELRC and RS.

frequent node changes, as it effectively reduces the burden on the blockchain network.

VII. IMPLEMENTATION AND EVALUATION

We have demonstrated through theoretical analysis that GELRC possesses storage efficiency, local repair capability, and decoding fault tolerance. Furthermore, we conducted simulation experiments to evaluate its fault tolerance, local repair capability, and the proposed group coding exchange method. We simulated a blockchain network on Tendermint, which is an open source permissioned blockchain system. We simulated a maximum of 40 nodes and used a separate device to simulate node operations for encoding and decoding data to compare LRB, BFT-Store and SeF.

After considering the research conducted by Liu et al. [53] on the selection of IoT devices, we select a device with Intel Core i7-7700 CPU, 4 cores, 3.6 GHz, 8GB RAM and 512GB disk space to simulate the entire encoding and decoding process of a node.

A. Fault Tolerance of Decoding

In the first experiment, we will compare the fault tolerance between our LRB architecture with BFT-Store. We compare how many errors or losses they can tolerate when decoding. We select LRB using (12, 9, 2, 6)-GELRC and BFT-Store using (9, 6)-RS to compare their probability of successful decoding under different numbers of lost nodes, as shown in Fig. 6. As we can see in Fig. 6, when the number of lost nodes is less than $n-k$ ($n-k = 3$ in Fig. 6), both of them can guarantee the success of decoding. However, when the number of lost nodes continues to increase, BFT-Store cannot complete decoding, while LRB can still complete decoding with a certain probability when a small number of additional nodes are lost.

Therefore, as shown in the experimental results, LRB can achieve the same fault tolerance as BFT-Store under normal conditions. Additionally, under specific conditions, as analyzed in Section VI-C, LRB still has a probability of successful decoding before the number of fault nodes exceeds $n - k + \lfloor \frac{k}{r+1} \rfloor$. In conclusion, LRB has a stronger fault tolerance capability compared to BFT-Store.
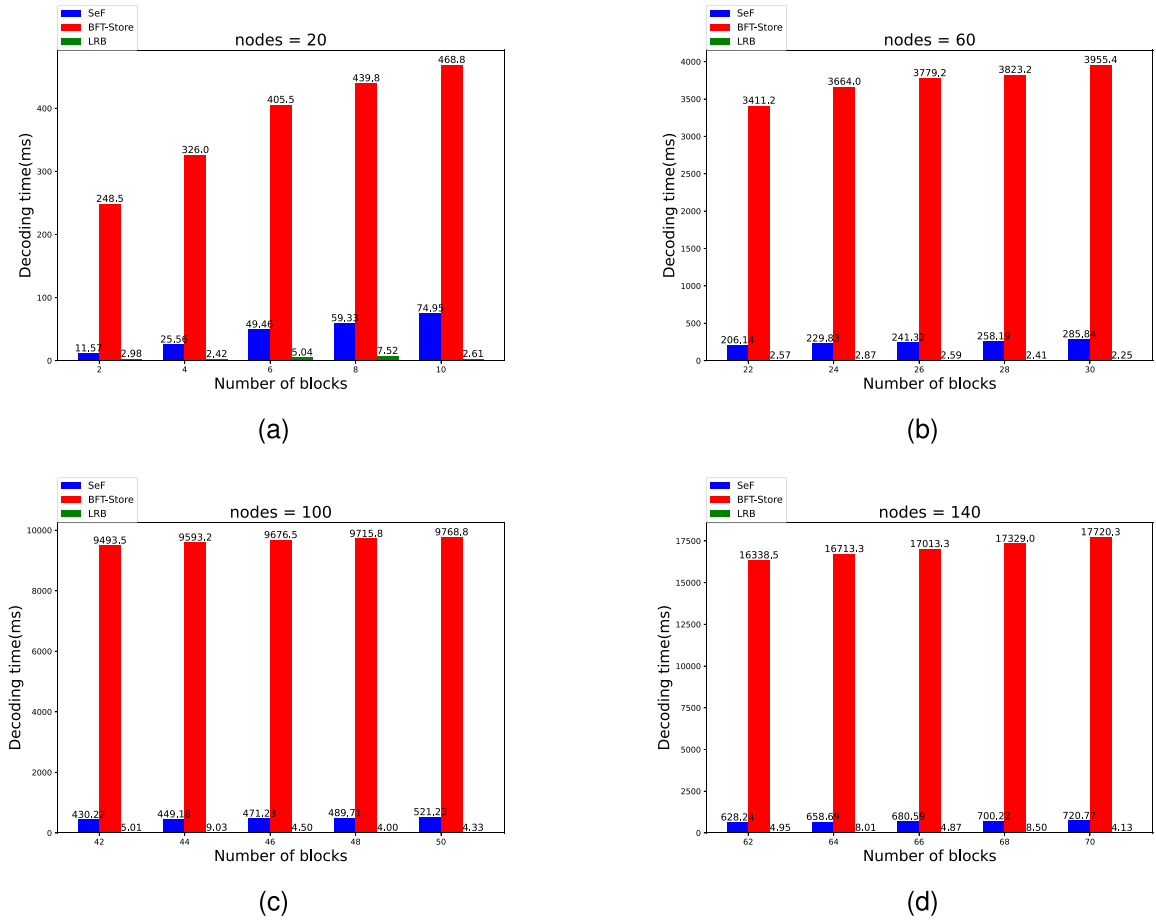
Fig. 7. Comparison of the decoding time of SeF, BFT-Store and LRB (a) $n = 20$. (b) $n = 60$. (c) $n = 100$. (d) $n = 140$.

B. Local Repair Capability

In experiment 2, we compared the local repair ability of SeF, BFT-Store and LRB. We selected four scenarios with node counts of 20, 60, 100 and 140. Starting from the original data block l being 2, 22, 42 and 62, respectively, we increment the number of blocks and compare the decoding time and bootstrap cost required for repairing single-point errors under these conditions for the three approaches.

Fig. 7 presents a comparison of the decoding time for SeF, BFT-Store and LRB as the number of original data blocks increases under four different scenarios. Additionally, Fig. 8 compares their bootstrap costs. To better illustrate the changing trends of local repair decoding time for the three architectures with increasing node counts and original data blocks, we selected specific data from Fig. 7 to construct Fig. 9.

Based on Fig. 7 and Fig. 9, we can see that the decoding time for BFT-Store and SeF increases significantly as the number of nodes and original data blocks grows. However, SeF demonstrates a much faster decoding speed compared to BFT-Store, while the decoding time variation of LRB is not significant. Because RS encoding used by BFT-Store involves matrix operations, its complexity increases as the original data and additional redundant data from node increases. This higher complexity leads to slower decoding speed. Both LT code used by SeF and the GELRC used by LRB employ XOR operation, which significantly reduces the decoding complexity. However,

since LRB achieves local repair using intra-group nodes, its bootstrap cost is much lower than SeF. Consequently, LRB requires far fewer XOR operations for decoding compared to SeF, resulting in faster decoding speed.

According to Fig. 8, we can compare the bootstrap costs of the three schemes. In order to ensure high-probability decoding, the bootstrap cost of SeF will be greater than the number of original data blocks, while the bootstrap cost of BFT-Store is the same as the number of original data blocks. The bootstrap cost of LRB is significantly lower than the former two, because the single-point repair of LRB only needs to contact the member nodes in the local repair group.

In summary, compared to SeF and BFT-Store, LRB performs better in handling single-point errors. It achieves local repair with lower bandwidth and computational costs at very low bootstrap cost.

C. Group Coding Exchange Method

In experiment 3, we use the traditional LRCs coding method to compare with GELRC, which utilizes the group coding exchange method. As mentioned above, the group coding exchange method makes each node only need to perform RS encoding once in each round of coding to reduce the encoding time, and the number of times RS encoding is performed in the traditional LRCs coding method is generally determined by the encoding parameter r. Therefore, as shown in Table I, we
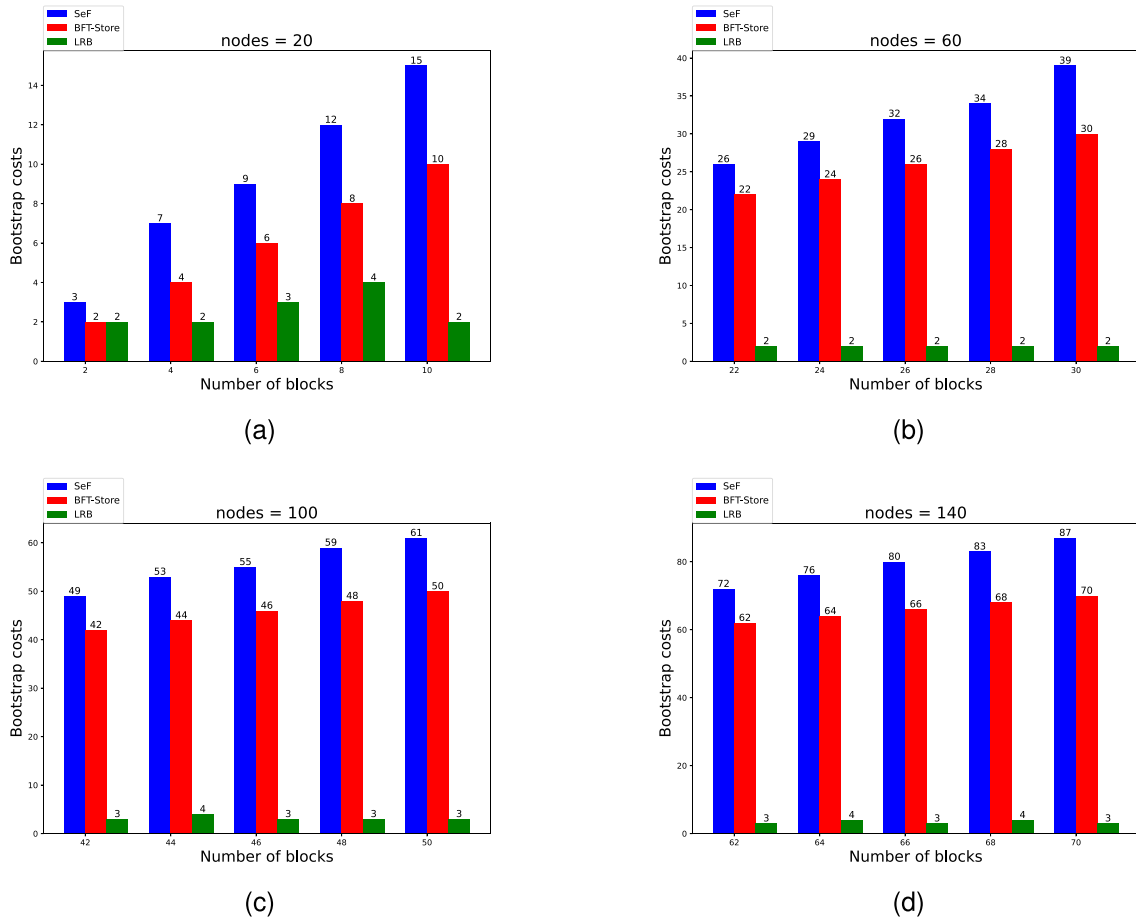
Fig. 8. Comparison of the bootstrap cost of SeF, BFT-Store and LRB (a) $n = 20$. (b) $n = 60$. (c) $n = 100$. (d) $n = 140$.
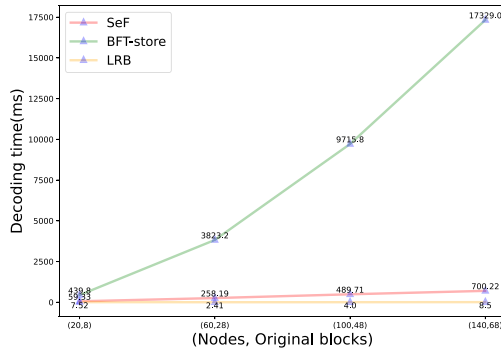

Fig. 9. Comparison of decoding time with increasing node counts and original data blocks.

TABLE I
THE SELECTION OF EXPERIMENTAL PARAMETER

Simple	L	n	r
1	12	9	2
2	21	12	3
3	32	15	4
4	45	18	5
5	60	21	6

conduct experimental comparisons on the following 5 sets of data. In order to better reflect the advantages of group coding exchange method, the parameter r in the data group increases sequentially.

Fig. 10(a) shows that with the increase of parameter r, the number of groups of original data blocks increases, which leads to an increase in the number of times a single node needs to perform RS encoding, so the encoding time required by a single node increases very rapidly. However, the node using GELRC only needs to perform RS encoding once, and its encoding time increases mainly because of the increase of the original data. Therefore, the encoding speed of GELRC

is significantly faster than that of traditional LRCs encoding method.

As mentioned above, due to the adoption of RS cods as pre-coding in GELRC, the complexity of RS coding increases significantly with the number of nodes and the number of original data blocks, resulting in a slower encoding speed. Fig. 10(b) illustrates the bandwidth required for the group coding exchange method to achieve the same encoding speed as single-node coding with lower computational complexity as the complexity of RS pre-coding increases. As shown in Fig. 10(b), at lower encoding complexity, our group coding exchange method may require greater bandwidth support to achieve the same encoding speed as single-node coding. In this case, if bandwidth resources are limited, we can consider using traditional LRCs architecture. However, as the speed of RS coding decreases, the advantages of our method will become apparent. At the same time, the main purpose of the group coding exchange method is to reduce the computational
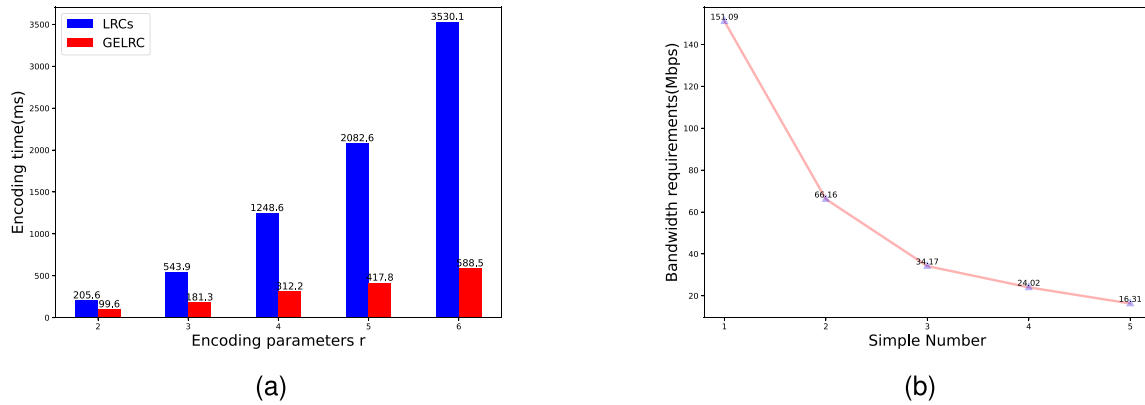
Fig. 10. Comparison of encoding time and bandwidth requirements between traditional LRCs framework and group coding exchange method (a) Encoding time. (b) Bandwidth requirements.

resource consumption of a single node, and as the parameter r increases, the savings in computational resources with our method will become more pronounced.

In conclusion, GELRC, which combines group coding exchange method and LRCs, effectively reduces the coding complexity for individual nodes while preserving the local repair capabilities of LRCs.

VIII. CONCLUSION

To address the challenges of high single-point repair costs, limited scalability, and reduced availability in existing coded blockchain technologies for integrating IoT and permissioned blockchains, we have introduced the LRB architecture. LRB combines LRCs with blockchain to achieve substantial storage savings and efficient local repairs while ensuring complete data recovery. The LRB architecture employs a node selection allocation algorithm that adapts to the dynamic IoT environment by choosing coding parameters and assigning local repair groups based on the blockchain network's current state. To overcome the high computational complexity associated with traditional LRCs, LRB incorporates GELRC, which integrates the group coding exchange method, significantly reducing encoding complexity for each node. Additionally, LRB eliminates the need for all blockchain nodes to re-encode when a new node is added.

However, there is still potential for further enhancement of the proposed approach. On one hand, future work will focus on improving data validation during the group coding block transmission process to better defend against sophisticated adversaries and enhance security against attacks. On the other hand, we will investigate decentralized storage solutions, specifically exploring methods for estimating the number of nodes in a blockchain network within a decentralized or asynchronous environment without admission control. This will allow us to dynamically adjust the selection of erasure code parameters, making our architecture more suitable for permissionless blockchains and achieving a higher level of decentralization.

REFERENCES

[1] M. A. Uddin, A. Stranieri, I. Gondal, and V. Balasubramanian, "A survey on the adoption of blockchain in IoT: Challenges and solutions," *Blockchain, Res. Appl.*, vol. 2, no. 2, 2021, Art. no. 100006.

[2] S. Smith, "IoT connections to reach 83 billion by 2024, driven by maturing industrial use cases," *Accessed*, vol. 10, p. 2021, Apr. 2020.

[3] S. Qi, Y. Lu, Y. Zheng, Y. Li, and X. Chen, "Cpds: Enabling compressed and private data sharing for Industrial Internet of Things over blockchain," *IEEE Trans. Ind. Informat.*, vol. 17, no. 4, pp. 2376–2387, Apr. 2021.

[4] D. Guru, S. Perumal, and V. Varadarajan, "Approaches towards blockchain innovation: A survey and future directions," *Electronics*, vol. 10, no. 10, p. 1219, 2021.

[5] C. Zhang, Z. Ni, Y. Xu, E. Luo, L. Chen, and Y. Zhang, "A trustworthy industrial data management scheme based on redactable blockchain," *J. Parallel Distrib. Comput.*, vol. 152, pp. 167–176, Jun. 2021.

[6] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *Int. J. Web Grid Serv.*, vol. 14, no. 4, pp. 352–375, 2018.

[7] N. M. Kumar and P. K. Mallick, "Blockchain technology for security issues and challenges in IoT," *Procedia Comput. Sci.*, vol. 132, pp. 1815–1823, Jun. 2018.

[8] N. K. Akrasi-Mensah et al., "An overview of technologies for improving storage efficiency in blockchain-based IIoT applications," *Electronics*, vol. 11, no. 16, p. 2513, 2022.

[9] J. Zhang, S. Zhong, J. Wang, and L. Wang, "An systematic study on blockchain transaction databases storage and optimization," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl., Big Data Cloud Comput., Sustain. Comput. Commun., Soc. Comput. Netw. (ISPA/BDCloud/SocialCom/SustainCom)*, 2020, pp. 298–304.

[10] A. Sward, I. Vecna, and F. Stonedahl, "Data insertion in bitcoin's blockchain," *Ledger*, vol. 3, pp. 1–23, Apr. 2018.

[11] X. Qi, Z. Zhang, C. Jin, and A. Zhou, "BFT-Store: Storage partition for permissioned blockchain via erasure coding," in *Proc. IEEE 36th Int. Conf. Data Eng. (ICDE)*, 2020, pp. 1926–1929.

[12] H. Wu, A. Ashikhmin, X. Wang, C. Li, S. Yang, and L. Zhang, "Distributed error correction coding scheme for low storage blockchain systems," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7054–7071, Aug. 2020.

[13] R. K. Raman and L. R. Varshney, "Coding for scalable blockchains via dynamic distributed storage," *IEEE/ACM Trans. Netw.*, vol. 29, no. 6, pp. 2588–2601, Dec. 2021.

[14] N. Chawla, H. W. Behrens, D. Tapp, D. Boscovic, and K. S. Candan, "Velocity: Scalability improvements in block propagation through rateless erasure coding," in *Proc. IEEE Int. Conf. Blockchain Cryptocurr. (ICBC)*, 2019, pp. 447–454.

[15] L. Zhang, T. Wang, and S. C. Liew, "Speeding up block propagation in bitcoin network: Uncoded and coded designs," *Comput. Netw.*, vol. 206, Apr. 2022, Art. no. 108791.

[16] M. Braun, A. Wiesmaier, N. Alnahawi, and J. Geißler, "On message-based consensus and network coding," in *Proc. 12th Int. Conf. Netw. Future (NoF)*, 2021, pp. 1–9.

[17] J. S. Plank, "A tutorial on reed–solomon coding for fault-tolerance in RAID-like systems," *Softw., Pract. Exp.*, vol. 27, no. 9, pp. 995–1012, 1997.

[18] W. E. Ryan, "An introduction to LDPC codes," *CRC Handbook Coding Signal Process. Rec. Syst.*, vol. 5, no. 2, pp. 1–23, 2004.

[19] S. Kadhe, J. Chung, and K. Ramchandran, "SeF: A secure fountain architecture for slashing storage costs in blockchains," 2019, *arXiv:1906.12140*.

[20] J. Singh, A. Banerjee, and H. Sadjadpour, "Secure and private fountain code based architecture for blockchains," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2022, pp. 1521–1526.

[21] L. Quan and Q. Huang, "Transparent coded blockchain," in *Proc. 15th Int. Conf. Emerg. Netw. Exp. Technol.*, 2019, pp. 12–13.

[22] B. Sasidharan and E. Viterbo, "Private data access in blockchain systems employing coded sharding," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2021, pp. 2684–2689.

[23] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 10, pp. 5843–5855, Oct. 2014.

[24] Q. Chen, C. Tang, and Z. Lin, "Compartmented secret sharing schemes and locally repairable codes," *IEEE Trans. Commun.*, vol. 68, no. 10, pp. 5976–5987, Oct. 2020.

[25] Q. Chen, C. Tang, and Z. Lin, "Locally repairable codes with heterogeneous locality constraints," in *Proc. IEEE Inf. Theory Workshop, (ITW)*, Guangzhou, China, 2018, pp. 1–5.

[26] J. I. Kokkala, D. S. Krotov, and P. R. Östergård, "On the classification of MDS codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 12, pp. 6485–6492, Dec. 2015.

[27] F. Mattoussi, V. Roca, and B. Sayadi, "Complexity comparison of the use of Vandermonde versus Hankel matrices to build systematic MDS reed-solomon codes," in *Proc. IEEE 13th Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, 2012, pp. 344–348.

[28] A. Tiwari and V. Lalitha, "Secure raptor encoder and decoder for low storage blockchain," in *Proc. Int. Conf. Commun. Syst. Netw. (COMSNETS)*, 2021, pp. 161–165.

[29] P. Zhang, Z. Liu, M. Zhou, and B. Huang, "A group-based block storage model with block splitting and unit encoding for consortium blockchains," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 3, pp. 2886–2897, Sep. 2023.

[30] S. S. Chow, Z. Lai, C. Liu, E. Lo, and Y. Zhao, "Sharding blockchain," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Soc. Comput. (CPSCom) IEEE Smart Data (SmartData)*, 2018, p. 1665.

[31] S. Das, A. Kolluri, P. Saxena, and H. Yu, "On the security of blockchain consensus protocols," in *Proc. 14th Int. Conf. Inf. Syst. Secur. (ICISS)*, 2018, pp. 465–480.

[32] M. Luby, "LT codes," in *Proc. 43rd Annu. IEEE Symp. Found. Comput. Sci.*, 2002, p. 271.

[33] R. K. Raman and L. R. Varshney, "Dynamic distributed storage for blockchains," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2018, pp. 2619–2623.

[34] V. Gugueoth, S. Safavat, S. Shetty, and D. Rawat, "A review of IoT security and privacy using decentralized blockchain techniques," *Comput. Sci. Rev.*, vol. 50, Nov. 2023, Art. no. 100585.

[35] S. Brotsis, N. Kolokotronis, K. Limniotis, and S. Shiaeles, "On the security of permissioned blockchain solutions for IoT applications," in *Proc. 6th IEEE Conf. Netw. Softw. (NetSoft)*, 2020, pp. 465–472.

[36] P. Thantharate and A. Thantharate, "ZeroTrustBlock: Enhancing security, privacy, and interoperability of sensitive data through zerotrust permissioned blockchain," *Big Data Cogn. Comput.*, vol. 7, no. 4, p. 165, 2023.

[37] D. Luo, Q. Cai, G. Sun, H. Yu, and D. Niyato, "Split-chain-based efficient blockchain-assisted cross-domain authentication for IoT," *IEEE Trans. Netw. Service Manag.*, vol. 21, no. 3, pp. 3209–3223, Jun. 2024.

[38] C. Feng, B. Liu, Z. Guo, K. Yu, Z. Qin, and K.-K. R. Choo, "Blockchain-based cross-domain authentication for intelligent 5G-enabled Internet of Drones," *IEEE Internet Things J.*, vol. 9, no. 8, pp. 6224–6238, Apr. 2022.

[39] X. Chen, S. He, L. Sun, Y. Zheng, and C. Q. Wu, "A survey of consortium blockchain and its applications," *Cryptography*, vol. 8, no. 2, p. 12, 2024.

[40] X. Fan, B. Niu, and Z. Liu, "Scalable blockchain storage systems: Research progress and models," *Computing*, vol. 104, no. 6, pp. 1497–1524, 2022.

[41] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proc. IEEE*, vol. 99, no. 3, pp. 476–489, Mar. 2011.

[42] C. Cachin and S. Tessaro, "Asynchronous verifiable information dispersal," in *Proc. 24th IEEE Symp. Rel. Distrib. Syst. (SRDS)*, 2005, pp. 191–201.

[43] C. Yang, A. Ashikhmin, X. Wang, and Z. Zheng, "Rateless coded blockchain for dynamic IoT networks," 2023, *arXiv:2305.03895*.

[44] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2551–2567, Jun. 2006.

[45] X. Li et al., "Repair pipelining for erasure-coded storage: Algorithms and evaluation," *ACM Trans. Storage (TOS)*, vol. 17, no. 2, pp. 1–29, 2021.

[46] J. Huang, L. Kong, G. Chen, L. Cheng, K. Wu, and X. Liu, "B-IoT: Blockchain driven Internet of Things with credit-based consensus mechanism," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2019, pp. 1348–1357.

[47] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman, "Solida: A blockchain protocol based on reconfigurable Byzantine consensus," 2016, *arXiv:1612.02916*.

[48] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," Cryptol. ePrint Arch., IACR, Bellevue, WA, USA, Rep. 2016/917, 2016.

[49] V. Gramoli, "From blockchain consensus back to Byzantine consensus," *Future Gener. Comput. Syst.*, vol. 107, pp. 760–769, Jun. 2020.

[50] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[51] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 931–948.

[52] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2018, pp. 583–598.

[53] Z. Liu et al., "PPRU: A privacy-preserving reputation updating scheme for cloud-assisted vehicular networks," *IEEE Trans. Veh. Technol.*, early access, Dec. 8, 2023, doi: 10.1109/TVT.2023.3340723.

Zihan Jiang received the B.S. degree from the Jiangsu University of Technology in 2022. He is currently pursuing the M.S. degree with the Institute of Artificial Intelligence, Guangzhou University. His research interests include coding theory, blockchain, and applied cryptography.

Qi Chen received the B.S. degree in mathematics from the University of Information Engineering in 2001, the M.S. degree in mathematics from the National University of Defense Technology in 2006, and the Ph.D. degree in mathematics from Guangzhou University in 2011. He has been with Guangzhou University since 2017. He is also a Visiting Researcher with the Lion Rock Labs of Cyberspace Security, HKCT. His research interests include secret sharing, cryptography, coding theory, and blockchain.
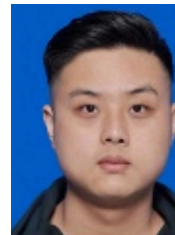
Zhihong Deng received the M.S. degree from the School of Mathematics and Computational Science, Hunan University of Science and Technology in 2021. He is currently pursuing the Ph.D. degree with the School of Mathematics and Information Science, Guangzhou University. His research interests include blockchain, applied cryptography, and algorithmic game theory.

He Zhang received the M.S. degree in mathematics from the Nanjing University of Aeronautics and Astronautics in 2019, and the Ph.D. degree in mathematics from Guangzhou University in 2024, where he is currently a Postdoctoral Fellow. His research interests include coding theory, finite fields and their applications, and cryptography.