Secure and Trust-Oriented Edge Storage for Internet of Things

Junxu Xia*, Geyao Cheng*, Siyuan Gu*, Deke Guo*†, *Senior Member, IEEE*

***Abstract*—The edge storage is a promising paradigm to support the Internet of Things (IoT) data storage, and is more efficient than the cloud storage in terms of the bandwidth overhead, the response latency, and so on. However, existing edge storage models cannot offer the security-aware data robustness and the adaptable data sharing among many uncertain users, due to the limitations of the utilized fault-tolerant storage technologies and the access control methods. In this paper, we propose a secure and trust-oriented edge storage model, which would efficiently tackle the aforementioned two challenging issues in the IoT environment. More precisely, we first propose a Robust and Secure Edge Storage model, RoSES, using the Totally Local Reconstruction Code (TLRC) method presented in this paper. It can achieve data robustness, high security, and lightweight computation at end devices. We further propose a Trust-Oriented Data Access (TODA) strategy for our RoSES model, which supports a wide and adaptable range of legitimate data accesses from uncertain requesters for IoT data sharing. We conduct extensive comparison and simulations to evaluate the performance of our new edge storage models. The results show that our model can efficiently realize the data storage, data recovery, and data sharing at the network edge, saving about 35% of storage cost and 76% degraded read latency. Besides, the data leakage probability is significantly reduced during the data storage and sharing processes.**

***Index Terms*—Internet of Things, Trust, Edge Storage, Security.**

I. Introduction

INTERNET of Things (IoT) is a collective set of technologies that connects and organizes a network of large-scale lightweight devices. Over the recent years, IoT technologies have gained more attention to realize essential services in many fields, such as the smart city [1], the smart healthcare [2], and the smart transportation [3], etc. With the dramatically increasing deployment of IoT devices, massive sensitive data have been generated at high speed, which brings the "Big Edge Data" challenge. The emerging edge storage systems [4]–[7] bring an efficient solution for tackling this challenge. It enables the quick and scalable response of big data by provisioning sufficient resources at network edge on demand. Meanwhile, with the support of the edge storage systems, more IoT devices can be deployed to improve the Quality of Services (QoS). This further exacerbates the "Big Edge Data" challenge and puts higher requirements for the edge storage systems.

*Science and Technology on Information Systems Engineering Laboratory National University of Defense Technology, Changsha Hunan 410073, P.R. China.

†College of Intelligence and Computing, Tianjin University, Tianjin 300350, P.R. China.

E-mail: junxuxia@gmail.com, chenggeyao13@nudt.edu.cn, gusiyuan18@nudt.edu.cn, guodeke@gmail.com.

*Deke Guo is the corresponding author.

In order to store the massive data, a large number of distributed edge servers would participate in the edge storage system to provide higher storage capacity. However, the vast number of edge servers brings significant challenges to the design of edge storage systems. First, it is very challenging to design a robust edge storage system over many distributed edge servers, each of which exhibits less reliability than the cloud. In reality, server failure is very common in the edge storage system. Thus, the edge storage system always pursues efficient fault-tolerant technologies to keep data availability when facing server failures. Second, the edge storage system consists of a large number of individual edge servers and cannot ensure that the stored data will not be leaked by the owners of each edge server. Third, data producers can also be data users in the IoT environment, and data sharing among uncertain users is an essential requirement for IoT applications. This requires an effective access control mechanism to achieve adaptable data sharing. In summary, it is essential to design a secure and trust-oriented edge storage system to meet the demands of IoT data.

Many edge storage systems [4]–[7] are designed to address the aforementioned problems. To keep the data availability, the existing edge storage systems [4]–[6] generally deploy multiple replicas of each file and store them in different edge servers. When an edge server fails due to disk damage, downtime, and so on, the service can be automatically switched to another replica to ensure the reliability of data storage. Other systems [7], [8] adopt the erasure coding methods to obtain lower storage space overhead than those replica-based methods. They encode the original file into multiple segments, which are distributed among many nodes, and retrieve a small portion of segments to reconstruct the original file. To protect the private data, the original file will be encrypted at the end devices first [9]. Then, replicas or encoded segments of the encrypted file will be transmitted to different edge servers for storage. For any file, only the owners of the secret key or some permitted requesters can access the encrypted replicas or segments and decrypt them via the interface provided by the edge storage system to realize the data sharing.

Despite such progress of edge storage systems, it remains open to tackle the *secure data robustness* problem and the *adaptable access control* problem. The first one is the *secure data robustness* problem. The data privacy protection via encryption often brings extra-high computation overhead, especially for the large-scale IoT data and lightweight end devices. Although the erasure code-based systems [7], [8] can gain higher data security because each storage node only holds a small portion of the original file, there is still a

potential risk of data leakage. The root cause is that some edge servers would access the entire data in the process of storage or encoding, which is not conducive to data security. The second one is the *adaptable access control* problem. Providing a virtualized interface to the owners of the secret key for accessing data would significantly limit the large-scale data sharing, especially for the IoT environment with many uncertain users. It is a more flexible access control scheme via evaluating the trust levels of data requesters by the data owner or even the third-party evaluation agents [10]. However, the agents are semi-trusted that may disobey the agreement in their business interests, leading to the leakage of private data. Furthermore, the agents conduct trust-level evaluations of the data requesters based on the registration information and historical logs, thereby only work for registered data users.

In this paper, we propose our secure and trust-oriented edge storage model to tackle the challenges above. To realize *secure data robustness*, we present the Robust and Secure Edge Storage model, RoSES, based on our new-designed Totally Local Reconstruction Code (TLRC). Our RoSES can achieve data robustness with lightweight computation of end devices, and each edge server is prevented from obtaining the complete file throughout the storage cycle to guarantee the data security. Thereafter, we further improve the RoSES with *adaptable access control*, which introduces the Trust-Oriented Data Access strategy, TODA, for adaptable data sharing. Our TODA strategy is adaptable enough for both the semi-trusted third-party evaluation agents and the unregistered data users, thus supporting a wide and adaptable range of legitimate data accesses in the data-sharing environment. Our major contributions include:

- We propose a new variant of erasure codes, i.e., the Totally Local Reconstruction Code (TLRC), to achieve secure data robustness for edge storage. Compared to prior erasure codes, our TLRC permits that: 1) no storage server has the opportunity to access all data segments even if recovering failed segments, which strengthens the protection of data privacy; 2) only a small portion of segments are required to generate redundant segments, which saves the communication overhead greatly.
- We propose the Robust and Secure Edge Storage model, RoSES, based on our TLRC. The coding structure of TLRC enables our RoSES model to realize data security even without encryption. Furthermore, we migrate the computation of redundant segments to the edge, thus, our RoSES can achieve lightweight computation of end devices.
- We propose the Trust-Oriented Data Access strategy, TODA, for our RoSES model. Our TODA strategy can support a wide and adaptable range of legitimate data accesses from uncertain data requesters, even if 1) the information of those requesters is not enough for the data owner or some evaluation agents to conduct trust-level evaluations; 2) some evaluation agents are bribed to provide wrong checks for malicious data requesters.

The rest of this paper is organized as follows. Section II reviews the related work. Section III provides a problem statement. In Section IV, we present a detailed description of the proposed RoSES model and the TODA strategy. Section V reports our experimental results. Finally, Section VI concludes this paper.

II. Related Work

In this section, we introduce the related works in terms of data robustness and access control in the edge storage systems.

A. Data Robustness in Edge Storage Systems

To mitigate the impact of server failures such as disk damage, downtime, and so on, edge storage systems need to adopt fault-tolerant mechanisms to keep the stored data available. Many existing edge storage systems [4]–[6] generally deploy multiple replicas of each file and store them in different edge servers to guarantee data robustness. When a storage node fails, the service can be automatically switched to another replica to ensure the reliability of data storage. However, maintaining replicas in such systems can be prohibitively expensive because of extra-high storage space overhead. Furthermore, when one of the storage servers is attacked, the entire data can be easily leaked out, which is not conducive to the protection of private data at the edge. Although some of the private data may be encrypted in advance, in addition to the extra-high computation overhead, the possibility of the encryption key being cracked cannot be completely ruled out. Therefore, we should prevent malicious users from getting the entire data or its ciphertext fundamentally.

Compared with replica-based storage systems, erasure code-based storage systems [7], [8] can significantly save the storage space and avoid entire data leakage, because each storage node only keeps a small portion of the original file. However, servers may not necessarily follow the established rules to protect data privacy during the recovering process in the complex edge environment. For example, when a segment becomes unavailable due to server failures, the degraded read operation will be initiated on a new storage node to recover the failed segment through extracting all the involved segments. If the new storage node is malicious, this node can reconstruct the original file via these involved segments, thus leading to privacy data leakage. Therefore, it is a challenging work to achieve secure data robustness, which can avoid the storage node to access the entire data.

In this paper, we propose a new variant of erasure code, i.e., Totally Local Reconstruction Code (TLRC). In this fault-tolerant coding scheme, the data recovery can be completed on a group scale, and thereby no server can access all of the data segments to generate the entire message during the storage and recovery process. Based on TLRC, our RoSES model can achieve both secure data robustness and lightweight of end devices.

B. Access Control in Edge Storage Systems

In many existing secure edge storage models [4] [5], only the key owners or specifically assigned users can access the stored data via the virtualized interface, which greatly limits

the IoT data sharing. Several methods aim to achieve more flexible data access control. In the Access Control List (ACL) based solutions [11] [12], the data owner would first specify an access control list for the data, and only the requesters in the list can access the data using their secret keys. In the Role-based Access Control (RBAC) methods [13], only the requesters with the appropriate roles can decrypt the ciphertext. The access control schemes based on Attribute-Based Encryption (ABE) [14] [15] are proposed for controlling data access based on attributes, thus enhancing the flexibility of data access.

Critically, most of those existing schemes cannot support data access control for new requesters, especially in the IoT environment, where the population flow is rapid and unpredictable. This fact dramatically influences the practical deployment of the existing schemes. To facilitate data sharing between different data users, it is more flexible to conduct access control via the trust-oriented evaluation by the data owner or the trusted third-party agents, based on the established rules [10]. However, 1) the third-party agents should be semi-trusted, because some may be bribed or make wrong checks in practice; 2) the data requester may not have registered at the third-party agent, thus cannot get the comprehensive and rational reputation evaluation.

In this paper, our TODA strategy adopts several agents for trust evaluations. Each agent independently manages the data access control of one edge server. In this case, each agent only has the right to control the partial data access of one edge server, which can decrease the data leakage risk caused by bribery. In addition, the data requester can access the data legitimately even if he only registers at some of the involved third-party agents. This enlarges the range of data users and facilitates data sharing in the IoT environment.

III. Problem Statement

A. Design Challenges

1) Why we prefer edge storage for IoT data: In cloud-based storage architectures, massive IoT data will be transmitted to the remote centralized data center for storage and analysis via the Internet. This undoubtedly incurs high expensive communication overhead and long communication latency, especially when the transmission volume is tremendous. As a result, when the data center suffers from intensive data requests, or under attack [16], the service of a large number of users cannot be guaranteed. Further, the centralized service providers may violate data privacy policies to trade-sensitive data without notifying the data owner, which impacts the data security and privacy. The decentralized edge storage can address the aforementioned drawbacks, with sufficient storing/computing resources for short-term IoT data storage. However, there are still many challenges waiting to be further exploited.

2) Challenges on data storage at the network edge: The edge computing environment collaborates with both the private clouds and business servers. Therefore, the edge storage incurs less reliability of the data storage compared with the cloud, resulting in common server failures. In order to support the data robustness when some storage servers fail, the replication or erasure code-based scheme has been widely utilized. However, the existed fault-tolerant storage models have the potential risk of data leakage because some edge servers would access the entire data in the process of storage or encoding. This is not conducive to the protection of private data. In addition, the data encoding process is quite time-consuming and computation-intensive, especially when the real-time IoT data is generated at high speed on lightweight devices. Therefore, secure data robustness and lightweight should be further addressed to meet the edge storage requirements.

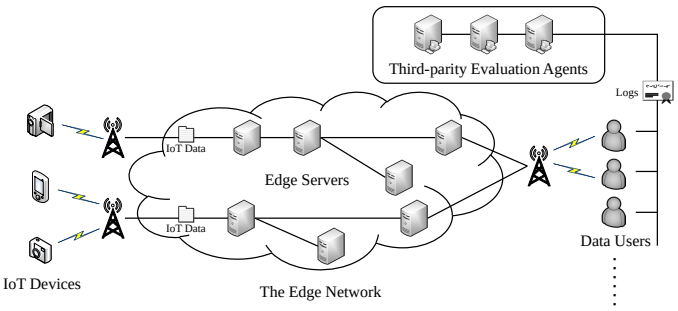


Fig. 1. The overview of our RoSES model.

3) Challenges on data access control at the network edge: To facilitate data sharing, the data owner would like to assign the trustworthy data requesters to access data. When he is off-line or unavailable to judge the trustworthiness of the data requesters, he could delegate some trusted third-party agents to evaluate the trust level of data requesters. If the evaluated trust level is over the pre-defined threshold, the requester can access the data. The evaluation results are generated based on the registration information as well as the historical data request logs at the third-party agents. In this case, there are two limitations for the trust-oriented access control: 1) the third-party agents are semi-trusted, that some may be bribed or make wrong checks in practice; 2) the data requester may not register at the third-party agent, thus limiting the data-sharing range. Therefore, how to ensure trust-oriented and adaptable data access is challenging work.

In this paper, we consider such a system involving four different kinds of entities, as illustrated in Fig. 1. The *IoT devices*, such as cameras and sensors, generate the real-time data at high speed, and send the data to the edge storage systems. The *edge servers* are collaborated with private clouds and business servers, which provide an open storage system with affluent bandwidth resources and storage/computation capability. The *data users* interact with edge servers for consuming various services (e.g., data storage and data access). The user can be a data owner or a data requester. The data owner can access the data directly, while the data requester should be first certified with its trust level before obtaining the access rights. The *third-party evaluation agents* have functions and capability to provide trust-level evaluations for data requesters who have registered with submitting their relevant certificates or proof materials.

B. Design Goals

To provide secure storage and adaptable access for IoT data at the network edge, our designed system should achieve the

following performance goals:

- **Lightweight**: It should alleviate the computation and storage burdens on the end devices. To be specific, the end devices only generate, divide, and send the data message to the edge servers. Complex computing tasks, such as data encoding and encryption, should be avoided, or liberated from the end devices.
- **Robustness**: It should keep data availability. When some servers fail, the stored data should still be available, or the lost data can be recovered.
- **Security**: It should achieve data security. We should prevent that any edge server has the opportunity to access the entire data message, for fear that the private data is leaked by the server owners or attackers. The access control mechanism should be elaborately designed to prevent malicious users from obtaining the stored data.
- **Adaptability**: It should support a wide and adaptable range of legitimate data accesses from various data requesters, even if 1) the information of those requesters is not enough for the data owner or some evaluation agents to conduct trust-level evaluations; 2) some evaluation agents are bribed to provide wrong checks for malicious data requesters.
- **Trustworthiness**: It should achieve legitimate data access control for the trustworthy data requesters, even if introducing the semi-trusted third-party agents.

IV. The Robust and Secure Edge Storage with Adaptable Access Control

In this section, we first propose a new variant of erasure code, i.e., Totally Local Reconstruction Code (TLRC). Based on that, we present the Robust and Secure Edge Storage model (RoSES). Thereafter, we improve the RoSES model with Trust-Oriented Data Access (TODA) control for large-scale IoT data sharing.

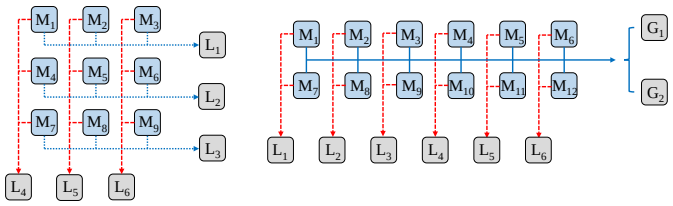
A. Totally Local Reconstruction Code

Recovering failed segments with traditional erasure codes generally carries a significant risk of data leakage, because the server needs to extract the entire data to generate the global parity segments. In this case, malicious servers can easily obtain the entire data when recovering failed segments, especially at edges, where various edge servers join the edge storage system automatically. To this insight, we propose a new variant of erasure codes, Totally Local Reconstruction Code (TLRC), to achieve data robustness with higher security.

TLRC only generates local parity segments, which are computed based on a portion of data segments to avoid data leakage. Specifically, for the coding structure TLRC(k,r,l) with l rows and r columns, the original data message is firstly divided into k data segments where $k{=}r{\times}l$. There are k/r data segments in each column, and k/l data segments in each row. The data segments in each row or column are in one group. Each local parity segment is computed based on the data segments in each group. The encoding operation retrieves the data segments from different locations and generates l horizontal, r vertical local parities, respectively. That is, each horizontal local parity is generated by extracting the r data segments in each row and performing an XOR operation on them, while the generation of each vertical local parity segment requires l data segments being transferred in a column. The total number of segments (data and parity) of TLRC(k,r,l) is $n{=}k{+}r{+}l$.

TABLE I
Summaries of Applied Keys

Keys	Description
M	The complete IoT data message
$M_1, M_2, \ldots$	The separated segments of the IoT data M
CM_i	The encrypted segments of the IoT data M
L	The local parity segment computed based on a group of data segments
G	The global parity segment computed based on all data segments
k	The number of divided data segments of message M
m, l, r	The number of computed parity segments of message M
K	The generated data encryption key
CK	The encrypted data encryption key
$K_1, K_2, \ldots$	The partial key
CK_i	The encrypted partial key
pk	The public key
sk	The secret key
$NoRC$	The number of chosen reputation centers
AA	The data access policy AA is generated with regard to public reputation threshold, based on personal interactions and experiences (e.g., in social networking activities)
$rk_{RC_i \to u'}$	The re-encryption key to re-encrypt a ciphertext computed by RC_i's public key into the one that can be decrypted by the secret key of user u'



(a) TLRC (9, 3, 3) (b) LRC (12, 6, 2)

Fig. 2. An illustrative comparison between the TLRC$(9,3,3)$ and LRC$(12,6,2)$.

The recovery operation of TLRC is invoked on segment failures. When a data/parity segment fails, other involved segments will be transferred to a new server to recover this failed segment. The operation can be completed on the group scale without contacting segments in other groups. Recovering a single failure in any data or parity segment requires only l segments transferring from the same column or r segments transferring from the same row. In this way, no server would contain the information of all data segments, which decreases the communication overhead in the process of segment transmission and strengthens privacy protection.

Our TLRC achieves these advantages with slightly losing the MDS properties, which means that the coding scheme should tolerate arbitrary $n{-}k$ server failures [17] [18]. In contrast, our TLRC can only tolerate arbitrary 2 failures, and specified 3 or more failures in certain circumstances. Specifically, for any failure node in the TLRC scheme, if there is another or more failures in its located row as well as column, the recovery is failed. Therefore, our TLRC scheme can recover the data with tolerating 3 failed segments, except for the situation where there are exactly one horizontal local parity failure, one vertical local parity failures, and the data

segment failure at the cross-location of the local parities (such as L_3, L_6, M_9 in TLRC$(9,3,3)$ in Fig. 2). The reason for this recovery failure is that the local parity segment can only be recovered by the data segments in its group. When that failed data segment cannot be recovered, the entire recovery would fail. This property also supports the deployment of third-party evaluation agencies in Section. IV-C2, thus, we can use fewer agencies to achieve legitimate data access control.

Generally, the theory probability of data un-recoverability of arbitrary 3 failed nodes ($P_{TLRC}(u=3)$) in TLRC(k,r,l) is shown as follows:

$$P_{TLRC(k,r,l)}(u=3) := \frac{k}{C^3_{k+r+l}}.$$

Our scheme can reconstruct the data with at most $l+r-1$ segment failures. As the coding structure TLRC$(9,3,3)$ shown in Fig. 2 (a), when the data segments M_1, M_2, M_3, M_4, and M_7 fail, the M_4 and M_7 can be recovered with segments $\{M_5, M_6, L_2\}$ and $\{M_8, M_9, L_3\}$, respectively. Thereafter, the M_1, M_2, M_3 can be recovered in the same way. In the actual application scenario, such as Facebook's data warehouse and other production HDFS clusters, over 98% of failure modes require recovery of a single segment, another 1.87% have two segment failures, and just less than 0.05% appear three or more segment failures [19] [20]. Therefore, our TLRC has excellent abilities in data recovery for most of the segment failure situations.

Compared with the Local Reconstruction Codes (LRC), which is adopted in Microsoft Azure Storage [21], Facebook HDFS-Xorbas [22], etc., TLRC incurs less transmission cost in the process of parity computation. For example, as Fig. 2 shows, TLRC$(9,3,3)$ and LRC$(12,6,2)$ (a fast coding way for the LRC in Microsoft Azure Storage presented in literature [23]) contain the same proportion of redundant parity segments, which are $\frac{l+r}{k+l+r}=\frac{3+3}{9+3+3}=0.4$ and $\frac{l+r}{k+l+r}=\frac{6+2}{12+6+2}=0.4$, respectively. However, there are $k\times 2=9\times 2=18$ data segments being transmitted to compute the 6 local parities in TLRC, while LRC requires $k\times 3=12\times 3=36$ segments to compute the 8 parities. In the process of degraded read, recovering a lost data segment or local parity only requires the segments being transferred from its group using TLRC, while all data segments are required to be transferred for recovering the global parities (G_1, G_2) in the LRC.

In addition, no server could contact all data segments during the entire storage cycle with our TLRC scheme, which protects the data privacy satisfactorily. Other schemes do not have this property due to the process of constructing or recovering the global parities via communicating with all data segments. In terms of recoverability, the theoretical probability of un-recoverability for arbitrary 3 failed nodes is $P_{TLRC(9,3,3)}(u=3) := \frac{9}{C^3_{9+3+3}} = 1.978\%$, while that of LRC$(12,6,2)$ is $P_{LRC(12,6,2)}(u=3) := \frac{6C^1_2}{C^3_{12+2+6}} = 1.052\%$. In summary, TLRC is efficient enough to decrease the communication overhead between edge servers and reduce leakage risk of private data, at the expense of slightly more unrecoverable probability.

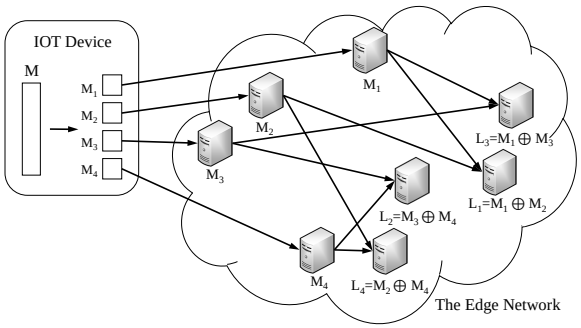


Fig. 3. An illustrative example of the robust and secure edge storage model based on the TLRC code.

B. The Robust and Secure Edge Storage Model Based on TLRC

Our Robust and Secure Edge Storage model, RoSES, is designed based on the TLRC scheme, which is customized for IoT devices to achieve secure data robustness in the decentralized edge storage. To be specific, 1) TLRC does not need to compute global parities. Thus, no edge server has the opportunity to contact all data segments, which strengthens the data security and privacy protection; 2) the local parities are computed based on a portion of data segments. Hence, the bandwidth overhead of both data construction and failure recovery can be efficiently decreased.

In addition to the data robustness and security, another distinguished design characteristic of our RoSES model is lightweight for end devices. We propose that the sophisticated generation of parity segments is accomplished at edge servers instead of the end devices. This practice can efficiently save the precious computing resources of end devices and fully exploit the computing and bandwidth resources of edge servers.

To meet the requirement for lightweight computation of end devices, parity segments are computed at edges, while the end device just needs to divide the raw data into multiple data segments and sends them to different edge servers for storage. Each edge server receives a data segment from the end device, then transfers it to a sub-level server according to the structure of the utilized TLRC scheme. Thereafter, each sub-level server generates a local parity segment with the received data segments from the up-level servers. In this way, the data is satisfactorily coded into data/parity segments and stored at the network edge. Note that, due to the characteristic of TLRC, only a small fraction of data is extracted for a server to generate a parity segment or recover a failed segment. Thus, the complete data is not available to any server during the entire storage cycle, which further strengthens the security of data storage.

Fig. 3 shows an illustrative example of our RoSES model using TLRC$(4,2,2)$. The original message M is first divided into 4 data segments, i.e., M_1, M_2, M_3, and M_4, each of which is transferred and then stored at a nearby edge server. Thereafter, based on the network topology and the generation structure of TLRC, M_1 and M_2 would be transferred to a same sub-level edge server for computing a local parity (L_1) through XOR operation, and so as M_1 and M_3, M_2 and M_4, M_3 and M_4.

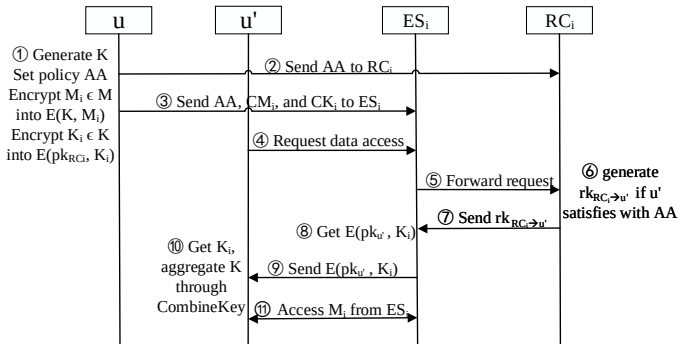


Fig. 4. The information flow of the trust-oriented data access control when the data owner is unavailable.

The metadata would be stored in the data center for convenient overall management and schedule.

In practical applications, the value of parameter l and r of TLRC(k,r,l) should be similar. This could avoid the unbalanced communication overhead and information access between different edge servers, thus, provide better network performance and data security.

C. The Improved RoSES Model with the Trust-Oriented Data Access Control

To facilitate the data sharing between different data users, we improve the RoSES model with Trust-Oriented Data Access (TODA) control. Our TODA is tailored for the RoSES model, offering a more flexible and secure data sharing scheme based on the TLRC coding structure. Generally, the data owner can determine whether the data requester is trusted to access the data. However, when the owner is off-line or not capable to determine the trust level of the requester, data sharing will be greatly limited. Instead, with TODA, the third-party agents will be delegated for data access control. This provides a flexible access mode for urgencies without the real-time participation of the data owner. The core of TODA is to prevent the untrustworthy requesters, or even the malicious edge storage servers, from obtaining the original file during data sharing. In the following statement, we use a specific kind of third-party agents, Reputation Centers, to illustrate our model.

The Reputation Center (RC) is the party that manages the registration material (such as professional certifications) and historical behavior (such as the data request logs). Based on the managed information, the RCs can evaluate the trust level of the registered users. However, in practical, RCs should be semi-trusted because they may be bribed to make wrong checks for their own interests. In addition, RCs cannot evaluate the trust level of the un-registered data requesters. This limits the range of data-sharing users.

In our trust-oriented data access model, multiple third-party agents are included for trust-level evaluations, which can address the aforementioned limitations satisfactorily. In particular, our access control scheme can adapt the situations that the data requester is un-registered or some RCs are bribed, thus supporting a wide and adaptable range of legitimate data access in the data-sharing environment.

1) Information flow of the trust-oriented data access control: The information flow of the trust-oriented data access control falls into two categories: the data owner is available or unavailable. When the data owner u is available, the data access right is totally controlled by the data owner. If the evaluated trust level of the data requester is over the predefined threshold, the data owner would issue the secret key for data requester u' [14] [24]. When the data owner is unavailable, the reputation centers are responsible for the data access control. The information flow consists of 11 steps as shown in Fig. 4, including the stages of the *data storage* and the *data access*. To illustrate the information flow more clearly, we first present several involved fundamental functions as follows:

- $E(Key,P)$. This function describes that a plaintext P (partial message M_i or partial key K_i) is encrypted by the key Key (the generated encryption key K, or the public key of RC pk_{RC}).
- $REKeyGeneration(pk_{RC}, sk_{RC}, pk_{u'})$. This function ensures RC to generate an re-encryption key $rk_{RC\rightarrow u'}$, based on the public key of RC (pk_{RC}), the secret key of RC (sk_{RC}), and the requester's public key u' ($pk_{u'}$) [25].
- $RE(rk_{RC\rightarrow u'}, E(pk_{RC},P))$. This function transforms $E(pk_{RC},P)$ into $E(pk_{u'},P)$ using the re-encryption key $rk_{RC\rightarrow u'}$ [26].
- $DivideKey(K,h)$. This algorithm can divide the input key K into h parts, where $NoRC > h \geq 1$.
- $CombineKey(K_1,K_2,\ldots)$. The combine algorithm aggregates the partitioned key K_i into a complete key K.

The stage of *data storage* involves 3 steps, as Step $1 \sim 3$ shown in Fig. 4. The first step is data preparation. Specifically, the data owner u generates an encryption key K and the policy AA, where AA is related to the trust-level threshold for accessing data. Each data segment M_i would be encrypted into $E(K,M_i)$ by the generated encryption key K. The key K is also divided into segments via $DivideKey$ and then encoded by the erasure code to generate $NoRC$ separate keys. Each separate key K_i is encrypted into $E(pk_{RC_i},K_i)$ by the public key of RC_i. Thereafter, the data owner sends the data access policy AA to RCs (Step 2). Meanwhile, each encrypted data segment $CM_i = E(K,M_i)$ is transmitted to the edge servers, while each encrypted partial key $E(pk_{RC_i},K_i)$ is only transmitted to the edge servers controlled by RCs (Step 3). We called the edge server controlled by RCs as the *slave edge server*, which holds both CM_i and CK_i. Note that, our model can also protect data security even if the original data are not encrypted, as long as the reliable communication protocol ensures that the data will not eavesdrop during transmission. In this case, the workload of the end device can be further reduced.

The stage of *data storage* consists of 8 steps, as Step $4 \sim 11$ shown in Fig. 4. Specifically, when a data user u' requests the data access (Step 4), all involved edge servers would be informed based on the stored location information. The *slave edge servers* would further inform their corresponding RCs for trust-level evaluations, while other involved edge servers transfer segments to the data requester u' directly (Step 5). In Step 6, RCs would issue the re-encryption key $rk_{RC_i\rightarrow u'}$

through $REKeyGeneration(pk_{RC_i}, sk_{RC_i}, pk_{u'})$ for requester u' after the eligibility check (e.g., the trust-level of the requester is over the predefined threshold based on policy AA at the checking time). Thereafter, each RC would send its $rk_{RC_i \to u'}$ to the subordinate *slave edge server* (Step 7). For Step 8, the *slave edge server* would generate $E(pk_{u'}, K_i)$ through re-encrypting $E(pk_{RC_i}, K_i)$ by $rk_{RC_i \to u'}$. After *slave edge servers* send $E(pk_{u'}, K_i)$ to the data requester u' (Step 9), the requester can gain partial key K_i via his secret key $sk_{u'}$, and then gain the encryption key K by $CombineKey(K_1, K_2, \ldots)$ (Step 10). Finally, the data requester u' can obtain the data M through decrypting the ciphertext by the encryption key K. This method supports confidential data sharing without exposing the underlying message and users' secret keys.

2) The adaptation of the trust-oriented data access control: To adapt the situations when the data requester is un-registered or some RCs are semi-trusted, multi RCs are involved in our model for trust-level evaluations. To be precise, there are three operations of RCs when receiving the data request: 1) agree, if the requester's reputation is evaluated beyond the predefined threshold based on the policy AA; 2) reject, if the reputation of the requester is below the threshold; 3) moderate, if the data requester is not registered, so that the RC is not convinced enough to evaluate its trust level.

The RCs can decide whether to send the re-encryption key $rk_{RC_i \to u'}$ to the corresponding *slave edge servers* based on their trust-level evaluation results. Each RC only manages its own slave edge server, such that the RC is independent for data access control. Each slave edge server will only send data if its master RC agrees, otherwise it will not send. In this way, the requester can only receive the data from those edge servers whose master RCs agree. When the number of agreed RCs exceeds a certain threshold, the requester can decode the original file according to the TLRC coding scheme. Therefore, even if the requester is not registered at all the RCs, he can still access the data as long as a certain number of RCs believe that the requester is trustworthy, thus achieving the adaptable data access control.

In addition, some RCs may collude with the requester or be bribed, since those RCs are semi-trusted. To avoid this, we deploy a summary module before RCs send their trust-level evaluation results to the corresponding slave edge servers, as Fig. 5 shows. To be specific, when at least one RC judges that the request is illegal, the summary module will directly deny the access. Hence, no slave server can get permission for sending data, even if some RCs agree. In this way, the malicious requester still cannot get the data, although he bribes some RCs and gets their agreements.

Fig. 5 shows an example of the trust-oriented data access (TODA) control based on our RoSES model. We set exactly four RCs (RC_1, RC_2, RC_3, RC_4) to manage the data access via conducting trust-level evaluations, i.e., $NoRC$=4. In the data storage stage, the data owner u first divides the data message M into 4 data segments and then encrypts them using the generated encryption key K. The key K is divided into 3 separate keys via $DivideKey(K, 3)$, then coded into $NoRC = 4$ segments by erasure code $\mathrm{RS}(3,1)$ at the end device. Each coded key segment K_i is encrypted using RC_i's public key

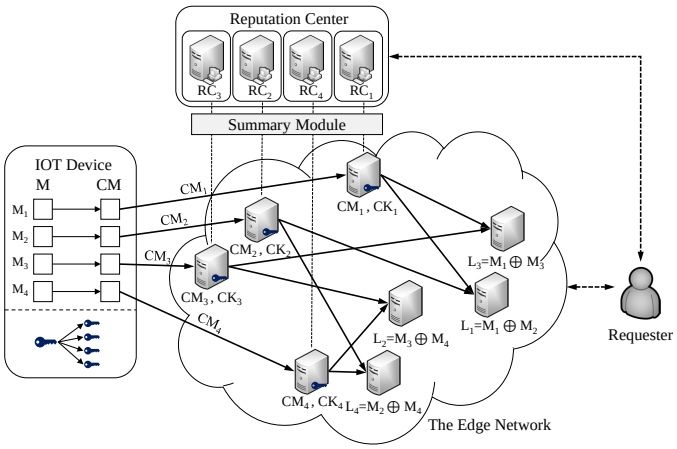


Fig. 5. An illustrative example of the TODA strategy based on the RoSES model.

pk_{RC_i}, respectively. All of these encrypted segments CK_i are transferred to the nearby edge servers.

The *slave edge servers* of the four RCs store the encrypted data segments CM_1, CM_2, CM_3, CM_4 and the separate keys CK_1, CK_2, CK_3, CK_4. In particular, when the number of data segments is beyond the number of RCs, i.e., $k > NoRC$ in $\mathrm{TLRC}(k, r, l)$, 4 RCs are still enough to manage the data access. The four RCs manages 4 data segments that form a rectangle in the TLRC coding scheme (e.g., M_1, M_2, M_4, M_5 in $\mathrm{TLRC}(9,3,3)$ in Fig. 2). Thus, as long as the slave edge servers do not transfer their segments, the rest segments on the other servers cannot reconstruct the original data. This is designed according to the aforementioned property of our TLRC scheme presented in Section. IV-A, that the original data cannot be reconstructed by TLRC when there are two failed segments in both a row and a column. With this feature, we can choose the edge servers with higher security as slave edge servers, which are managed by RCs directly. Thus, users cannot reconstruct the original data, even if they obtain all the data from the other edge servers that do not have master RCs. This further improves the data security of edge storage.

Note that, when the number of available RCs is less than 4, i.e., $NoRC < 4$, our TODA still works, with slightly lower data security. To be specific, the requester can reconstruct the original encrypted file based on the received segments, if any RC agrees to send data. However, the file still cannot be decrypted when the untrustworthy requester does not pass the trust-level evaluation of some RCs. The reason is that RCs would not send their re-encryption key $rk_{RC_i \to u'}$ to the data requester, unless he passes their trust-level evaluations. Therefore, the untrustworthy requester cannot know the contents of the file, even if he colludes with a part of RCs.

Furthermore, our TODA scheme ensures the data security-protection with or without data encryption. When a data user requests the stored data, there are two steps to access the message, i.e., ciphertext reconstruction and key reconstruction. Our model can also guarantee data security without data encryption, while the probability of data leakage is slightly higher (Section IV-C3). In contrast, if the file requires higher security, it should be pre-encrypted at the end device to achieve

more secure data access control. Besides, encryption can also avoid data leakage when the data is transmitted from the end devices to different edge servers via the network. Thus, the attacker can only get some encrypted segments, but cannot decrypt them to get the contents of the original file.

3) The data leakage probability with different number of chosen RCs: In the aforementioned trust-oriented data access control model, we set exactly four RCs for the data access through trust-level evaluations. Actually, more RCs can be utilized for trust-level evaluations to strengthen data security protection. Furthermore, the data can be un-encrypted if there is no privacy involved or the communication protocols can guarantee data security in the transmission process. We analysis the probability of data leakage when the data is encrypted and un-encrypted as follows:

Probability of data leakage when the data is encrypted: We let p refer to the probability that any RC concludes with other entities after bribery, and γ refer to the probability that a requester has registered at any RC. We assume that four RCs are utilized for trust-level evaluations. The original data is coded by TLRC$(9,3,3)$, and the encryption key is coded with RS$(3,1)$. In this way, the encrypted data can be reconstructed when one or more RCs agrees, but no RC rejects the data request. The key can be reconstructed only when no RC rejects and at least 3 RCs hold the attitude of agreement. Therefore, the leakage probability of encrypted data ($DL_e(4)$) can be summarized as:

$$DL_e(4)=p^4+C_4^3p^3(1-p)(1-\gamma). \tag{1}$$

This equation can be further expanded into the scenario with i RCs, where $i>4$. We assume the encryption key is coded with RS$(i-1,1)$. Thus, the leakage probability encrypted data with i RCs, $DL_e(i)$, is calculated as follows.

$$DL_e(i)=p^i+C_i^{i-1}p^{i-1}(1-p)(1-\gamma) \tag{2}$$

Probability of data leakage when the data is un-encrypted: The probability of data leakage with encryption is smaller than that without encryption. The reason is that the reconstruction of the encryption key is omitted for the un-encrypted messages, which weakens the data-security protection. The data leakage probability of un-encrypted message with four RCs, $DL_{ue}(4)$, is calculated as follows.

$$\begin{aligned}DL_{ue}(4)=&C_4^1p[(1-p)(1-\gamma)]^3+C_4^2p^2[(1-p)(1-\gamma)]^2\\&+C_4^3p^3(1-p)(1-\gamma)+p^4.\end{aligned} \tag{3}$$

This equation can also be expanded into the scenario with more RCs. Note that, with the property of TLRC, the data leakage probability would change with different number of RCs and even their corresponding locations. Considering the irregularity and complexity in the data leakage probability, we only give the data leakage probability of un-encrypted messages with 5 and 6 RCs ($DL_{ue}(5)$ and $DL_{ue}(5)$):

$$\begin{aligned}DL_{ue}(5)=&C_4^1p[(1-p)(1-\gamma)]^4+C_5^2p^2[(1-p)(1-\gamma)]^3\\&+C_5^3p^3[(1-p)(1-\gamma)]^2+C_5^4p^4(1-p)(1-\gamma)+p^5.\end{aligned} \tag{4}$$

$$\begin{aligned}DL_{ue}(6)=&\frac{4}{5}C_4^1p[(1-p)(1-\gamma)]^5\\&+[\frac{4}{5}(C_6^2-1)+\frac{1}{5}(C_6^2-3)p^2[(1-p)(1-\gamma)]^4\\&+C_6^3p^3[(1-p)(1-\gamma)]^3+...+p^6.\end{aligned} \tag{5}$$

V. Performance Emulation

In this section, we empirically evaluate the performance of our RoSES model with trust-oriented data access. At first, we introduce the experimental settings in Section V-A, including the topology construction, comparison algorithms, and evaluation metrics. Then we conduct performance emulations in Section V-B.

A. Experimental Settings

1) Topology construction: In our evaluations, we generate network topologies with the number of nodes ranging from 40 to 220. The topologies are constructed using random Erdős-Rényi graphs [27]. Each node in the topology can represent an edge server in our experiments. We assume the link cost, such as renting price, is proportional to its geographical distance. In the generated random graph, we embed the graph into a 25×25 grid uniformly and set the link costs equal to the corresponding distances. Furthermore, we set the bandwidth of each link in the generated topology as $1Gbps$ and the data volume of each message as $1GB$.

2) Comparison algorithms: We complemented and compared the following algorithms:

Local Reconstruction Codes (LRC). It is the fault-tolerant coding way adopted in Microsoft Azure Storage [21], Facebook HDFS-Xorbas [22], etc., to replace the traditional RS coding. LRC(12,2,2) (abbreviated as LRC2) is the general coding way adopted in Microsoft Azure Storage, while LRC(12,6,2) (abbreviated as LRC6) is a fast coding way designed for Microsoft Azure Storage, which is presented in literature [23]. In our evaluations, both the local and global parities of LRC are computed at edge servers to decrease computational pressure of IoT devices.

Totally Local Reconstruction Code (TLRC). In this algorithm, only local parities are generated based on the group of data segments in the same row or column. Note that, the value of rows and columns should be similar to avoid the unbalanced communication cost and storage overhead, thereby to provide better network performance. In this paper, we compare the performance between different TLRC coding structure: TLRC(4,2,2), TLRC(9,3,3), and TLRC(16,4,4), which are abbreviated as TLRC4, TLRC9, TLRC16, separately.

Note that, in our evaluations, we do not compare the performance of the traditional erasure codes. The reason is that all data segments should be involved in the generation of parity segments in the traditional erasure codes, which would consume a huge amount of bandwidth resources and incur a high potential risk of data leakage in our lightweight edge storage model.

3) Evaluation metrics: Before listing the evaluation metrics, we first introduce three operations of our model as follows.

Data Storage. In the data storage operation, the data message is divided and separately stored at several edge servers

in a distributed way. This process includes data segment transmission (the divided data segments are transferred from the end devices to the edge servers) and parity segment generation (the data segments are further transferred between edge servers to generate local/global parity segments).

Degraded Read. The degraded read operation is invoked by recovering the segment failures. In this process, the stored segments should be transferred for data recovery from their corresponding edge servers to the new-chosen servers. In our experiments, we evaluate the performance of degraded read for recovering one failure and two failures.

Data Access. For the data owner, the data can be accessed directly by the generated key. For the data requester, the encrypted message and the encryption key should be reconstructed to obtain the original message. The data leakage happens when a malicious data requester bribes enough number of RCs to obtain the original message through reconstructing the encrypted data and the encryption key.

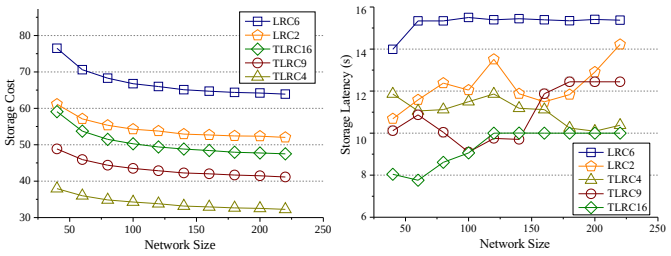
We rigorously analyze two performance metrics for the storage and degraded read operations of the RoSES model as follows.

- **Cost**. With the preset cost of each link in the generated topology, the storage/degraded read cost refers to the total cost of the transmission links in the data storage/degraded read operations. Note that, the transmission links in our evaluations are determined based on the minimum spanning tree [28].
- **Latency**. With the preset link bandwidth and message volume, the storage/degraded read latency refers to the transmission time for all involved separated segments arriving at their target edge servers through the minimal-cost links, in the storage/degraded read operations. Note that, the traffic congestion in some links would seriously impact the storage/degraded read latency.

The evaluation methodology is represented as follows. First, we generate the decentralized network topology as described in Section. V-A1. Thereafter, given an IoT data message, we randomly choose one node of the generated topology as the end device, and then the nearest nodes would be chosen as the storage edge servers. For the message storage operation, the separated data segments would be transmitted to the nearby chosen nodes through the constructed minimal spanning tree, aiming at minimizing the transmission cost. The data segments would further be transmitted to other nodes for generating parity segments, based on the given coding schemes. For the message recovery operation, new nodes would be chosen with the minimal accessorial cost for receiving all involved data/parity segments. Then, the failed segments would be reconstructed at the new nodes. In this case, the performance of data storage and data recovery in terms of cost and latency can be emulated separately. For the data access operation, we emulate the data leakage probability of encrypted/unencrypted data messages with a different number of RCs.

B. Performance Analysis

In this subsection, we conduct large-scale experiments to evaluate our RoSES scheme in the operations of data storage and data recovery. All of the experimental results are the average values of 100 repeated experiments on 100 different decentralized topologies.



(a) The average storage cost. (b) The average storage latency.

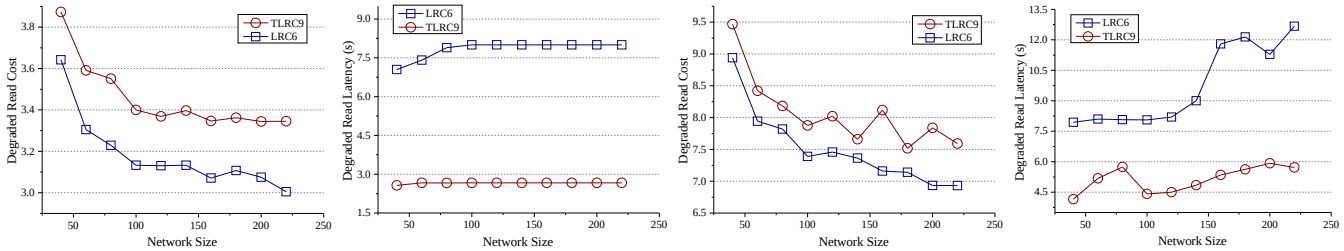
Fig. 6. The evaluations on the data storage between LRC and TLRC scheme.

1) Data storage: We emulate the data storage operation using four coding structures, i.e., LRC(12,2,2), LRC(12,6,2), TLRC(4,2,2), TLRC(9,3,3), and TLRC(16,4,4), in terms of storage cost and storage latency.

Fig. 6(a) plots the average storage cost with respect to the number of nodes in the generated decentralized topology. Although fewer segments are stored in LRC(12,6,2) and LRC(12,2,2) than that of TLRC(16,4,4), the LRC scheme consumes more storage costs. The reason is that all data segments should be transmitted for generating the global parity segments in the LRC scheme, while only a group of segments are needed for the local parity generation in the TLRC scheme. In addition, for the TLRC scheme, the storage cost would grow up with smaller granularity of data segments (the original message is separated into more data segments). The root cause is that more links would be occupied in the process of segment transmission when the original message is divided into more data segments. Furthermore, as the topology expands, the storage costs of all compared algorithms are gradually declined, because the minimal spanning tree would be generated with less cost when more alternative nodes are involved in the decentralized topology.

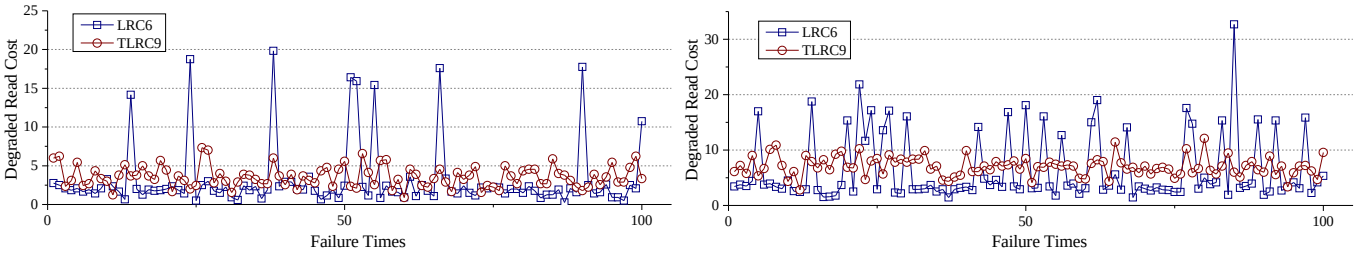
Fig. 6(b) plots the average storage latency in the storage process, with respect to the size of the generated topology. With the decrease of granularity of separated data segments in the TLRC scheme, shorter transmission latencies are achieved. The root cause is that, when the original IoT message is separated into data segments with smaller granularity, a larger minimal spanning tree is constructed with more chosen storage servers. Thus, the transmission volume would be spread out across the links, causing less traffic congestion and shorter latency. The chart fluctuates greatly for every involved coding structure. The reason is that, all transmission links are determined by the minimal spanning tree, with the optimization goal of reducing total transmission cost. Thus, traffic congestion may exist for the links with low cost, resulting in non-optimal and fluctuant storage latency.

2) Degraded read: Thereafter, we emulate the performance of the recovery consumption of one server failure and two server failures, separately, in terms of degraded read cost and degraded read latency. To reveal the essential differences of the LTC and TLRC schemes, we only compare the performance of TLRC(9,3,3) and LRC(12,6,2) in this subsection, which contain the same proportion of parity segments



(a) The average degraded read cost for recovering one server failure. (b) The average degraded read latency for recovering one server failure. (c) The average degraded read cost for recovering two server failures. (d) The average degraded read latency for two server failures.

Fig. 7. The evaluations on the degraded read operation between LRC(12,6,2) and TLRC(9,3,3).



(a) The degraded read cost per time in the same topology of 220 nodes. (b) The degraded read cost per time in the same topology of 220 nodes.

Fig. 8. The degraded read cost per time of LRC(12,6,2) and TLRC(9,3,3).

($\frac{l+r}{k+l+r}=\frac{3+3}{9+3+3}=\frac{6+2}{12+6+2}=0.4$).

Fig. 7(a) shows the average degraded read cost of LRC(12,6,2) and TLRC(9,3,3) in the process of recovering one server failure, with respect to the size of generated topology. The average degraded read cost of TLRC(9,3,3) is huger than that of LRC(12,6,2). The root cause is that there is only a small probability for recovering global parity segments that consumes huge degraded read cost in LRC(12,6,2). For the general data segments or local parity segments, the reconstruction can be completed on a group scale with only two segments being transferred to the new storage server. However, for the TLRC(9,3,3) scheme, all failures are recovered with three segments being transferred to the new storage server. Furthermore, the degraded read costs of all compared algorithms are gradually declined, as the topology expands. The reason is that the minimal spanning tree would be generated with less cost when more alternative nodes are involved in the decentralized topology.

Fig. 7(b) plots the average degraded read latency of LRC(12,6,2) and TLRC(9,3,3) in the process of recovering one server failure, with respect to the scale of the generated topology. The degraded read latency of LRC(12,6,2) is longer than that of TLRC(9,3,3). It is because that the traffic congestion for the links with low cost is more serious for the LRC(12,6,2) scheme, where all data segments should be transmitted for the reconstruction of global parity segments.

Fig. 7(c) shows the average degraded read cost of LRC(12,6,2) and TLRC(9,3,3) in the process of recovering two server failures, with respect to the number of nodes in the generated decentralized topology. The general trend is consistent with that for recovering only one server failure. However, the degraded read cost of both LRC(12,6,2) and TLRC(9,3,3) for recovering two server failures is more than twice as much as that for recovering only one server failure. The reason is that the optimal transmission paths based on the minimal spanning tree can be utilized for recovering one failure. However, there may be a server failure on the optimal paths for recovering a specific failure, when there are two failures waiting to be recovered. Thus, the sub-optimal transmission path would be employed, causing a little more degraded read cost.

Fig. 7(d) shows the average degraded read latency of LRC(12,6,2) and TLRC(9,3,3) in the process of recovering two server failures, with respect to the scale of generated topology. The increasing trend of degraded read latency is much more pronounced for recovering two failures than that of one failure in the same size of generated topology. The root cause is that, with the optimal spanning tree being constructed with more involved alternative nodes, the links with low cost may be utilized more exhaustedly, leading to a more sensitive increase of degraded read latency.

Fig. 8(a) plots the degraded read cost per time of LRC(12,6,2) and TLRC(9,3,3) for recovering one server failure in a generated topology with 220 nodes. It is obvious that the degraded read cost of LRC(12,6,2) vibrates with larger amplitude, compared with that of TLRC(9,3,3). The root cause is that the recovery of the global parity segment in LRC(12,6,2) would incur transmission of all data segments, while the reconstruction of any data segment or local parity segment can be completed on a group scale. For the recovery of segments in TLRC(9,3,3), the degraded read cost only fluctuates because both the data segments and the local parity segments can be recovered in a small scale, resulting in a similar cost.

Fig. 8(b) plots the degraded read cost per time of LRC(12,6,2) and TLRC(9,3,3) for recovering two server failures in the same topology of 220 nodes. The general trend is consistent with that for recovering only one server failure. The slight difference is that the degraded read cost of LRC(12,6,2) vibrates more frequently. The reason is that the probability of recovering global parity segments is larger when there are two

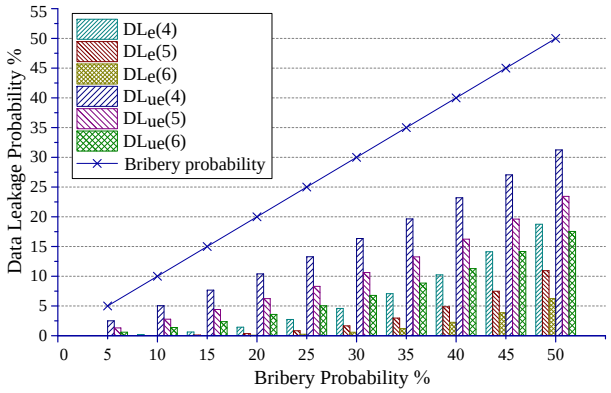


Fig. 9. The data leakage probability with different number of involved RCs.

node failures. Actually, the value of the peak point in Fig. 8(b) represents the degraded read cost for recovering exactly two global parity segments in LRC(12,6,2).

3) Data access: Finally, we emulate the data leakage probability with different bribery probability of RCs. The coding structure utilized for the stored data is TLRC(9,3,3), and the coding structure utilized for the encryption key is $\mathrm{RS}(i-1,1)$, where i represents the number of utilized RCs.

Fig. 9 plots the data leakage probability of encrypted/un-encrypted messages with a different number of involved RCs. The crossed line plots the bribery probability, which can represent the data leakage probability when only one RC is involved in the trust-level evaluation. The bribery probability is higher than the data leakage probability with more involved RCs, demonstrating the effectiveness of our data access control with multi RCs involved. Furthermore, the data leakage probability would decrease with more RCs being involved in the trust-level evaluations, because the malicious data requester should bribe more RCs to reconstruct the data message. The data leakage probability of the unencrypted message (DL_{ue}) is larger than that of the encrypted message (DL_e) with the same number of involved RCs. The root cause is that the reconstruction of the encryption key is omitted for the un-encrypted messages, which weakens the data-security protection.

In summary, our RoSES model with TODA control can conduct the data storage, recovery, and access tasks satisfactorily at the network edge. Specifically, with the same proportion of redundant segments, our TLRC coding can save about 35.4% of storage cost and 30.8% of storage latency. Further, in the process of data recovery of one server failure, our TLRC scheme can reduce degraded read latency by 76.6%, while causing about 8.70% more degraded read cost. In addition, the data leakage probability is significantly reduced with more RCs being involved in our TODA strategy.

VI. Conclusion

In this paper, we propose a robust and secure edge storage model with the trust-oriented data access control, which realizes the security-protected data robustness and adaptable sharing of IoT data in the decentralized edge storage systems. Specifically, to realize the security-protected data robustness, we propose a robust and secure edge storage model, using our new-designed totally local reconstruction code. It can achieve data robustness and higher security with lightweight computation at end devices. To realize the adaptable sharing of IoT data, the trust-oriented data access scheme is further proposed, which supports a wide and adaptable range of legitimate data accesses for IoT data sharing. The performance of our schemes is evaluated and justified through extensive comparison and simulations. The experimental results show the effectiveness of the secure and robust edge storage model and its improvement with adaptable access control.

Finally, we believe that IoT devices would invoke more urgent requirements for ultra-high bandwidth, ultra-low latency, and ultra-high privacy, with the advent of the 5G era. Thus, our edge storage model has a promising implementation prospect.

Acknowledgment

This work is partially supported by National Natural Science Foundation of China under Grant Nos. U19B2024 and 61772544, National key research and development program under Grant Nos. 2018YFB1800203 and 2018YFE0207600, Tianjin Science and Technology Foundation under Grant No. 18ZXJMTG00290.

References

[1] M. Shen, X. Tang, L. Zhu, X. Du, and M. Guizani, "Privacy-preserving support vector machine training over blockchain-based encrypted iot data in smart cities," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7702–7712, 2019.

[2] M. Shen, Y. Deng, L. Zhu, X. Du, and N. Guizani, "Privacy-preserving image retrieval for medical iot systems: A blockchain-based approach," *IEEE Network*, vol. 33, no. 5, pp. 27–33, 2019.

[3] M. Shen, J. Zhang, L. Zhu, K. Xu, and X. Tang, "Secure svm training over vertically-partitioned datasets using consortium blockchain for vehicular social networks," *IEEE Transactions on Vehicular Technology*, 2019.

[4] C. R. O. F. K. R. Neumann, Dirk; Bodenstein, "Stacee : enhancing storage clouds using edge devices," in *Proc.of ACM/IEEE Workshop on Autonomic Computing in Economics*, 2011, pp. 19–26.

[5] F. Rizzo, G. L. Spoto, P. Brizzi, D. Bonino, G. D. Bella, and P. Castrogiovanni, "Beekup: A distributed and safe P2P storage framework for ioe applications," in *Proc.of Conference on Innovations in Clouds, Internet and Networks, ICIN Paris, France, March 7-9*, 2017, pp. 44–51.

[6] A. Aral and T. Ovatman, "A decentralized replica placement algorithm for edge computing," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 516–529, 2018.

[7] H.-Y. Lin and W.-G. Tzeng, "A secure erasure code-based cloud storage system with secure data forwarding," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 6, pp. 995–1003, 2011.

[8] ——, "A secure decentralized erasure code for distributed networked storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 11, pp. 1586–1594, 2010.

[9] M. Shen, B. Ma, L. Zhu, X. Du, and K. Xu, "Secure phrase search for intelligent processing of encrypted data in cloud-based iot," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1998–2008, 2019.

[10] Z. Yan, X. Li, M. Wang, and A. V. Vasilakos, "Flexible data access control based on trust and reputation in cloud computing," *IEEE Transactions on Cloud Computing*, vol. 5, no. 3, pp. 485–498, 2017.

[11] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh, "Sirius: Securing remote untrusted storage," in *Proc.of the Network and Distributed System Security Symposium, San Diego, California, USA*, vol. 3, 2003, pp. 131–145.

[12] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc.of IEEE Symposium on Security and Privacy*, 2007, pp. 321–334.

[13] L. Zhou, V. Varadharajan, and M. Hitchens, "Achieving secure role-based access control on encrypted data in cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 12, pp. 1947–1960, 2013.

[14] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proc.of IEEE International Conference on Computer Communications, San Diego, CA, USA, March 15-19*, 2010, pp. 1–9.
[15] G. Wang, Q. Liu, J. Wu, and M. Guo, "Hierarchical attribute-based encryption and scalable user revocation for sharing data in cloud servers," *Computers & Security*, vol. 30, no. 5, pp. 320–331, 2011.
[16] L. Zhu, X. Tang, M. Shen, X. Du, and M. Guizani, "Privacy-preserving ddos attack detection using cross-domain traffic in software defined networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 628–643, 2018.
[17] I. Tamo and A. Barg, "A family of optimal locally recoverable codes," *IEEE Transactions on Information Theory*, vol. 60, no. 8, pp. 4661–4676, 2014.
[18] M. Sathiamoorthy, M. Asteris, D. S. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: Novel erasure codes for big data," *PVLDB*, vol. 6, no. 5, pp. 325–336, 2013.
[19] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A "hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers," in *Proc.of ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22*, 2014, pp. 331–342.
[20] ——, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the facebook warehouse cluster," in *Proc.of USENIX Workshop on Hot Topics in Storage and File Systems, San Jose, CA, USA, June 27-28*, 2013.
[21] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Proc.of USENIX Annual Technical Conference, Boston, MA, USA, June 13-15*, 2012, pp. 15–26.
[22] M. Sathiamoorthy, M. Asteris, D. S. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: Novel erasure codes for big data," *CoRR*, vol. abs/1301.3791, 2013.
[23] M. Xia, M. Saxena, M. Blaum, and D. A. Pease, "A tale of two erasure codes in hdfs," in *Proc.of USENIX Conference on File and Storage Technologies, Santa Clara, CA, USA, February 16-19*, 2015, pp. 213–226.
[24] Z. Wan, J. Liu, and R. H. Deng, "Hasbe: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 2, pp. 743–754, 2011.
[25] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Transactions on Information and System Security*, vol. 9, no. 1, pp. 1–30, 2006.
[26] M. Mambo and E. Okamoto, "Proxy cryptosystems: Delegation of the power to decrypt ciphertexts," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 80, no. 1, pp. 54–63, 1997.
[27] P. Erdős and A. Rényi, "On random graphs. i." *Publicationes mathematicae*, vol. 6, pp. 290–297, 1959.
[28] S. Pettie and V. Ramachandran, "An optimal minimum spanning tree algorithm," *Journal of the Acm*, vol. 49, no. 1, pp. 16–34, 2002.

Junxu Xia received the B.S. degree in management science and engineering from National University of Defense Technology, Changsha, China, in 2018. He is currently working towards the M.S. degree in the same department. His main research interests include data centers, cloud computing, and the distributed system.

Geyao Cheng received the B.S. degree in management science and engineering from National University of Defense Technology, Changsha, China, in 2017. She is currently working towards the M.S. degree in the College of Systems Engineering, National University of Defense Technology, Changsha, China. Her research interests include mobile computing and crowdsensing.

Siyuan Gu received the B.S. degree in mathematics from Officers College of PAP, Chengdu, China, in 2015. He is currently working towards the M.S. degree in College of Systems Engineering, National University of Defense Technology, Changsha, China. His research interests include edge computing and distributed computing.

Deke Guo received the B.S. degree in industrial engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001, and the Ph.D. degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 2008. He is currently a Professor with the College of System Engineering, National University of Defense Technology, and is also with the College of Intelligence and Computing, Tianjin University. His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks. He is a senior member of the IEEE and a member of the ACM.