Galaxy: A Scalable BFT and Privacy-Preserving Pub/Sub IoT Data Sharing Framework Based on Blockchain

Yuchao Zhang, *Member, IEEE,* Xiaotian Wang, Xiaofeng He, Ning Zhang, *Senior Member, IEEE,* Zibin Zheng, *Fellow, IEEE,* and Ke Xu, *Senior Member, IEEE*

***Abstract*—The emergence of the Internet of Things (IoT) technology in recent years has led to a considerable amount of data to be shared across different organizations. The publish and subscribe (Pub/Sub) paradigm, with its asynchronous, one-to-many and decoupling characteristics, is considered to be a promising communication model in IoT. However, designing a Pub/Sub framework for IoT data sharing confronts two challenges: Byzantine faults and privacy concerns. Byzantine nodes that are subjectively malicious or hacked by attackers may discard or forge data in the broker network composed of untrusted IoT organizations. Unauthorized brokers or clients may try to obtain the content of publications or subscriptions, thus violating the IoT data privacy. Existing works have limitations in terms of relatively low scalability and high overhead in tackling these two challenges. In this paper, we propose *Galaxy*, a blockchain based Pub/Sub IoT data sharing framework. To achieve Byzantine fault-tolerant (BFT) Pub/Sub, Galaxy adopts sharding to improve scalability and achieve efficient BFT Pub/Sub workflow within each shard with a novel leader rotation scheme. In attaining privacy-preserving Pub/Sub, a secret key sharing and encrypted Pub/Sub scheme is designed in Galaxy to achieve low overhead without breaking the decoupling of the system. We implemented a prototype of Galaxy and deployed it on Alibaba Cloud for experimental evaluation. The experiment results show the feasibility and efficiency of Galaxy.**

***Index Terms*—Internet of Things(IoT), Blockchain, IoT Data Sharing, Secure Communications, Security and Privacy.**

I. Introduction

With the rapid development of Internet of Things (IoT) technology in recent years, the number of IoT devices has grown dramatically and has been adopted in various fields, such as industrial manufacturing [1], healthcare [2], and smart grid [3], etc. These IoT devices and subsequent applications generate a massive amount of data that need to be collected, shared and analyzed across different organizations. In order to better meet the data sharing needs in the above scenario, some network protocols and data sharing systems based on the publish and subscribe (Pub/Sub) abstraction have been designed and used in IoT [4]–[6]. The Pub/Sub paradigm enables one-to-many, asynchronous, and decoupled communication between publishers and subscribers. Basically, there are three entities in a Pub/Sub system: brokers, subscribers, and publishers. Brokers are dedicated servers that provide pub/sub services. Subscribers subscribe to the topics they are interested in by sending subscriptions to brokers. Publishers advertise publications on specific topics to brokers, and these publications are matched against subscriptions and forwarded to interested subscribers by brokers. In the past years, services under the Pub/Sub abstraction have been widely used in various systems within datacenters like Apache Kafka [7], Pulsar [8] and Google Cloud Pub/Sub [9].

With the rise of IoT and edge computing, however, the broker network may be composed of edge servers controlled by untrusted tenants from different organizations. The lack of trust between organizations brings new challenges to the design of Pub/Sub system in this scenario: **Byzantine faults** and **privacy concerns**.

Byzantine faults: Pub/Sub systems often achieve fault tolerance and high availability by replicating data among brokers. Pub/Sub systems within data centers usually only consider crash fault [7], [10] and realize crash fault tolerance (CFT) based on consensus algorithms such as Paxos [11] and Raft [12]. However, in an IoT network composed of untrusted servers from different organizations, some data sharing involving business information may lead to conflicts of economic interests, resulting in Byzantine behaviors like message forgery and discarding. Additionally, brokers may be hacked by external attackers and thus may crash or become malicious, making the situation worse. Blockchain, a modern form of state machine replication (SMR) that achieves Byzantine fault tolerance (BFT), has attracted massive attention in recent years. Some works try to combine blockchain with Pub/Sub system to achieve a higher level of fault tolerance [6], [13], [14], while several issues still need to be further addressed: The high communication and cryptographic overhead of BFT consensus leads to its insufficient scalability in large-scale networks; some works only achieve incomplete BFT by adopting blockchain as an additional auditing system; hardware failures, software errors, and potential malicious attacks result in slow recovery of leader rotation in the leader-based BFT consensus

This work was supported by the Key Project of Beijing Natural Science Foundation under M21030.

Yuchao Zhang, Xiaotian Wang and Xiaofeng He are with the School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunication, Beijing, China (e-mail: {yczhang, wangxiaotian, xiaofenghe}@bupt.edu.cn).

Ning Zhang is with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, Canada. (e-mail: ning.zhang@uwindsor.ca).

Zibin Zheng is with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. (e-mail: zhzibin@mail.sysu.edu.cn).

Ke Xu is with the Department of Computer Science and Technology, Tsinghua University, Beijing, China (e-mail: xuke@tsinghua.edu.cn).

[15], [16], severely degrading the performance of the system. Therefore, a scalable BFT Pub/Sub system design is needed to ensure system safety and availability.

Privacy concerns: The information published in IoT contains highly privacy-sensitive data such as medical data and geographic information. Therefore, some publishers only want partial subscribers to see the content of specified publications. For instance, in a medical data sharing scenario, some patients would like to share their medical records only with designated hospitals. Similarly, subscribers may not want to disclose their personal interests, resulting in a desire to limit the exposure of their subscriptions in the network. Unfortunately, maintaining the privacy of both publishers and subscribers in the aforementioned trustless network is not possible without special efforts. Some existing works adopt attribute-based encryption (ABE) [17] to achieve privacy protection with access control [18], [19]. Due to the slow pairing-based computation used in ABE, it is not suitable for the frequent publish and subscribe operations in IoT. On the other side, symmetric encryptions are more efficient compared to ABE and other asymmetric encryption schemes. However, exchanging symmetric keys directly between publishers and subscribers would break the decoupling of the system, while relying on a trusted third party could result in a single point of failure. These issues calls for a privacy-preserving Pub/Sub system design with low overhead and without breaking the decoupling nature.

In this paper, we propose *Galaxy*, a scalable BFT and privcay-preserving Pub/Sub IoT data sharing framework based on blockchain. Galaxy adopts a two-layer architecture, comprising of a governance layer and a data layer. The governance layer, which is a a high-performance blockchain deployed in cloud and maintained by representative IoT organizations, provides global management services including cross-shard total order and secret key sharing. The data layer is a multi-shard blockchain system deployed on edge servers managed by different IoT organizations and provide clients with Pub/Sub service. To achieve **BFT Pub/Sub**, Galaxy adopts sharding, a widely used horizontal scaling technique, to improve scalability of the underlying blockchain based on a partially random shard assignment strategy. Within each shard, Galaxy achieves complete BFT workflow by embedding a streamlined BFT consensus module in each broker. To enhance the performance of the BFT consensus, a crash and withholding avoidance (CWA) leader rotation algorithm is proposed to avoid selecting failed nodes as leaders. To achieve **privacy-preserving Pub/Sub**, Galaxy adopts symmetric searchable encryption (SSE) [20] and advanced encryption standard (AES) [21] to reduce the cryptographic computation overhead, and uses threshold encryption [22] and access control list (ACL) to share the symmetric keys through the governance layer without destroying the decoupling nature of the system. We implement a prototype of Galaxy and deploy it on Alibaba Cloud for evaluation. The result shows that Galaxy achieves 3994 ops/sec and 3047 ops/sec with a total of 128 brokers (4 shards each with 32 brokers) under LAN and WAN settings respectively. Moreover, the proposed CWA algorithm outperforms the round-robin algorithm by a factor of 6x and 4x under the crash and withholding faults respectively when the proportion of the simulated malicious nodes is close to one third. Additionally, by comparing our privacy-preserving Pub/Sub scheme with a representative existing work [23], the results demonstrate our scheme is efficient and lightweight. In summary, our contributions are as follows:

1) *Trusted IoT Data Sharing Framework:* We propose Galaxy, a two-layer Pub/Sub IoT data sharing framework based on blockchain that enables trusted IoT data sharing across different organizations. Our framework is decoupled and modular, making it easily extendable and deployable on existing IoT systems.
2) *Scalable BFT Pub/Sub Design*: A partially random shard assignment strategy is used to improve the scalability of blockchain, a complete BFT Pub/Sub workflow is designed and a novel leader rotation algorithm is proposed to avoid electing a failed leader.
3) *Lightweight Privacy-Preserving Pub/Sub Design*: a secret key sharing and an encrypted Pub/Sub scheme is designed to achieve relatively low computational overhead and avoid breaking the decoupling of the system.
4) *Prototype Implementation and Evaluation*: We implemented a prototype of Galaxy and deployed it on Alibaba Cloud for evaluation. The results demonstrate the effectiveness and feasibility of Galaxy.

The remainder of this paper is organized as follows. Section II reviews related works. Section III defines the system model and gives an overview of Galaxy. We elaborate on the Galaxy's BFT Pub/Sub design in Section IV, followed by the privacy-preserving Pub/Sub design in Section V. In Section VI, we evaluate a prototype of Galaxy and analyze the experiment results. Finally, we conclude this paper in Section VII.

II. Related Works

In this section, we briefly review several related works on blockchain-based IoT data sharing, fault-tolerant Pub/Sub and privacy-preserving Pub/Sub systems.

A. Blockchain-based IoT Data Sharing

Due to the decentralization, traceability, and immutability of blockchain, researchers have proposed many IoT data sharing systems based on blockchain. Some works utilize the blockchain to authenticate the validity of the data or the clients. Shen *et al.* [24] proposed a blockchain-assisted device authentication scheme for cross-domain communication in industrial IoT. They design an identity management scheme to authenticate devices anonymously and a key agreement mechanism to negotiate session keys between devices. In [25], Fan *et al.* presented a secure blockchain-based scheme to ensure source data credibility in the fog environments and use the attribute-based signature to achieve fine-grained access control. He *et al.* [26] introduced a nested blockchain framework with dynamic node selection for IoT data storage and identity authentication. Some works also design encryption schemes together with blockchain to achieve data privacy. Zhang *et al.* [27] proposed a blockchain-assisted massive IoT data collection and management framework. They use a collaborative identity verification protocol to ensure reliable data source and a hierarchical data aggregation scheme to collect IoT data efficiently and securely. Qi *et al.* [28] proposed Cpds, a blockchain-based compressed and private data sharing framework for industrial IoT. They adopt a tree-based data compression method and an access control mechanism based on attribute encryption. Li *et al.* [29] proposed a privacy-preserving and rewarding data sharing scheme based on blockchain. They also use deniable ring signatures to ensure honest data users can refuse to be framed without revealing their true identity.

Different from the above work, Galaxy achieves BFT and privacy-preserving data sharing in a one-to-many, asynchronous, and decoupled manner based on the Pub/Sub communication paradigm and adopts a two-layer and modular architecture which is scalable and incrementally deployable.

B. Fault-tolerant Pub/Sub

Most existing Pub/Sub systems do not provide fault tolerance mechanism or only provide crash fault tolerance [4], [7]–[10]. For instance, Kafka [7], as a representative industrial Pub/Sub system, uses ZooKeeper [30] or KRaft (Raft [12] implementation in the new version of Kafka) to reach consensus on metadata among controllers. For the broker, Kafka allows passive replication within a replica group called partition. Systems like Kafka are designed for applications in the data centers owned by the same organization, and cannot be directly migrated to the scenario where brokers are deployed across different organizations and lack mutual trust.

BFT consensus and SMR algorithms applied in blockchain systems can be used to build decentralized trust in distributed systems. Some works try to ameliorate Byzantine faults in the Pub/Sub system by combining blockchain or traditional BFT-SMR [6], [13], [14], [31]. Ramachandran *et al.* [6] proposed Trinity, a system that combines MQTT broker [4] with existing blockchain including Ethereum [32], Hyperledger Fabric [33], IOTA [34] and Tendermint [35]. Their solution completely depends on the blockchain platform used, and only Tendermint can realize deterministic BFT without relying on digital currency. In [13], Huang *et al.* designed BPS, a blockchain-enhanced Pub/Sub system based on Kafka [7]. BPS stores Pub/Sub metadata and access control list into the blockchain to improve the security level of Kafka. This scheme only uses the blockchain as an auditing system and cannot guarantee the BFT of the Pub/Sub service itself. Similarly, HyperPubSub [14] uses Hyperledger Fabric [33] as an additional component for Kafka to build a decentralized Pub/Sub service and does not achieve BFT. Duan *et al.* [31] take BFT-SMaRt [36] as a module of brokers to realize complete BFT Pub/Sub. However, the scheme of running BFT consensus with high communication complexity among all brokers does not have sufficient scalability.

Different from the aforementioned works, Galaxy adopts a multi-shard architecture based on the high-performance streamlined BFT consensus with a novel leader rotation scheme to ensure both BFT and scalability of the system.

C. Privacy-preserving Pub/Sub

Existing works employ different cryptographic primitives to protect the privacy of pub-sub systems, but their schemes have different limitations in our scenario. ABE [17] is an encryption scheme that can realize attribute based access control and some works use ABE to achieve fine-grained privacy. Ion *et al.* [18] proposed a Pub/Sub system that achieves confidentiality and access control by combining ABE and proxy encryption [37]. Zhang *et al.* [38] use dual-policy ABE [39] to construct a privacy-preserving attribute-keyword search scheme for Pub/Sub systems on cloud platforms. The reason why ABE is not used in our work is that the relatively slow pairing-based cryptography computations used in ABE make it not suitable for frequent Pub/Sub operations in a practical system. On the contrary, symmetric encryption can provide higher efficiency under the risk of breaking decoupling. Crescenzo *et al.* [40] designed a three-party Pub/Sub protocol based on symmetric encryption and the third party should be honest in their scheme. Similarly, Rao *et al.* [41] proposed a filter privacy aware system based on the content-based Pub/Sub model, and a trusted anonymous engine is needed in the proposed system. In [42], Gaballah *et al.* designed an anonymous Pub/Sub protocol using distributed point functions and private information retrieval. However, their scheme requires relatively complex computations and relies on synchronicity assumptions, which is not practical in our scenario.Cui *et al.* [23] proposed a privacy-preserving Pub/Sub system to resist collusion attacks between brokers and clients. Their scheme adopts SSE and key-policy attribute based encryption (KP-ABE) scheme to achieve encrypted Pub/Sub matching and access control, which is similar to our work. However, their system is not totally BFT and relies on a centralized TA to distribute the symmetric key.

Based on the above works, Galaxy adopts searchable symmetric encryption in the Pub/Sub process to ensure low computational overhead and adopts a tag-based threshold encryption together with BFT consensus to share symmetric keys and avoid coupled key sharing or centralized key management.

III. Overview of Galaxy

In this section, we define the system model and give a high-level overview of Galaxy.

A. System Model

Galaxy assumes a partial-synchronous network model [43], where there exists an unknown Global Stabilization Time (GST), the network can behave asynchronously till GST. After GST, any message can be delivered within a bounded duration Δ. For both the governance layer and the data layer, Galaxy adopts the authenticated setting used in the permissioned blockchain, that is, there exists a Public Key Infrastructure (PKI) system and the public key of each node is known by all entities. Galaxy assumes that each client registers an identity within an organization and sends messages to nodes within his/her organization, so we do not consider DDoS attacks. The permissioned blockchain model is chosen by Galaxy for two reasons: compared with the public blockchain, the node identities are easier to manage and more suitable for the security needs of a block-based system for IoT. In addition, more efficient and deterministic consensus algorithms can be

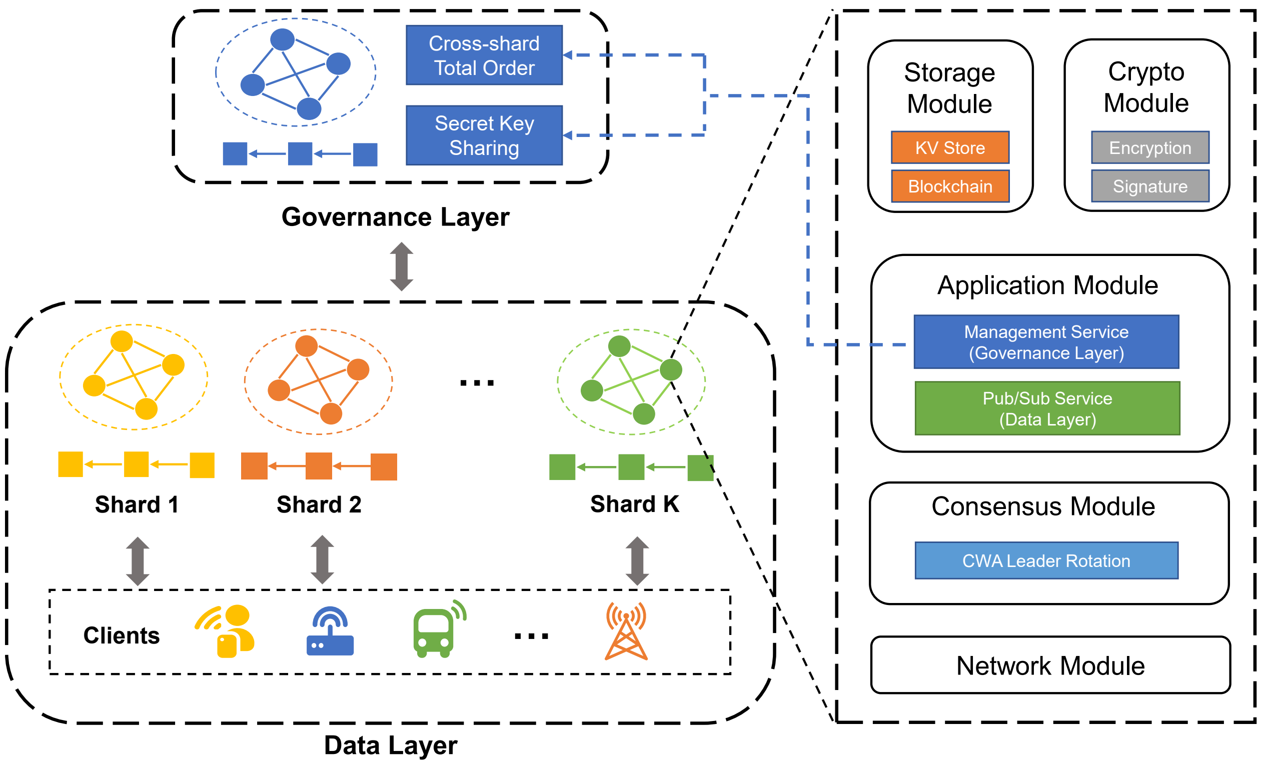


Fig. 1: An overview of Galaxy. The two-layer architecture of Galaxy (left), and the modular design of a Galaxy node (right).

adopted. However, the design of Galaxy can also be generalized to permissionless scenarios by incorporating hybrid consensus [44].

Galaxy assumes the nodes at the data layer are divided into S blockchain shard and each shard contains n_i ($1 \leq i \leq S$) nodes. Galaxy assumes the single blockchain at the governance layer contains m nodes. Galaxy assumes a Byzantine fault model, where at most f nodes within each shard or within the governance ledger are Byzantine nodes (we assume $f < \lfloor \frac{n_i}{3} \rfloor$ and $f < \lfloor \frac{m}{3} \rfloor$). Galaxy also assumes the communication channels between the honest nodes and clients are authenticated point-to-point channels and these channels are encrypted with TLS/SSL or other cryptographic protocol. Byzantine nodes may behave arbitrarily and collude with each other, but cannot compromise the communication channels between honest nodes and clients. Moreover, Galaxy assumes an adversary with polynomial time computing power can control faulty nodes and the adversary cannot break our cryptographic primitives.

B. System Overview

Galaxy consists of a governance layer and a data layer in a two-layer hierarchical structure, as shown in Fig.1. The data layer is a multi-shard blockchain network deployed on edge servers managed by different IoT organizations. Shards run in parallel and each shard has its own blockchain ledger. The data layer nodes implement BFT Pub/Sub service in the application layer and we refer to the data layer nodes as brokers. The Pub/Sub service in Galaxy is designed based on the topic-based and push-based Pub/Sub model, and only considers online clients connected to brokers, which is similar to MQTT [4]. Each subscription contains one or more specific topics of interest to a client. Each publication consists of a header and a payload, and the header contains one or more topics for matching with the subscriptions. When a broker receives a new publication, it matches the subscription stored locally and pushes the matching publication to all subscribers interested in the related topics. Unlike the multi-shard blockchain in the data layer, the governance layer consists of a high-performance permissioned blockchain with smaller network size called governance ledger, which is deployed in the data centers within cloud. The nodes in the governance layer are maintained by a few representative IoT organizations and management services including secret key sharing and cross-shard message total-order are implemented at the application module.

Each Galaxy node achieves functional decoupling through a modular design, as depicted on the right of Fig.1: The network module is responsible for the communication between node and clients, as well as between nodes; the consensus module implements BFT consensus and provide interfaces not tied to specific consensus algorithms for the application module; the application module implements specific application services by calling the interface provided by the consensus module, which is the main difference between the data layer and the governance layer; the storage module implements the storage of the latest data (we implement it as a key-value store) and the append-only blockchain ledger; The crypto module implements the encryption and signature primitives required by other modules, including the threshold encryption and the

SSE schemes used in our privacy-preserving Pub/Sub design. Galaxy's modular architecture enables the modules to be easily replaced, extended, and allows for incremental deployment on existing IoT frameworks. More implementation details are discussed in the Section IV.A. Next, we briefly introduce the BFT Pub/Sub and privacy-preserving Pub/Sub design of Galaxy respectively.

BFT Pub/Sub Design: Galaxy adopts sharding, a horizontal scaling technology widely used in distributed systems, to improve the scalability of the data layer. Due to the limitation of the number of malicious nodes in BFT consensus, our sharding strategy not only adopts randomness but also considers the relationship between organizations to better ensure the safety of each shard. For each broker in the data layer, we have implemented a complete BFT Pub/Sub workflow based on the BFT consensus and provided a series of simple APIs for clients. The implementation of our consensus module is based on the streamlined BFT consensus Hotstuff [45] to ensure $O(n)$ message complexity and optimistic responsiveness. In addition, we implemented the leader rotation algorithm as an independent function in the consensus module and propose a *crash and withholding avoidance (CWA)* leader rotation scheme instead of the traditional round-robin scheme in order to avoid selecting the failed nodes as leaders, and thus improve the performance of consensus. More details of our BFT Pub/Sub design are explained in Section IV.

Privacy-Preserving Pub/Sub Design: To achieve data invisibility to both brokers and unauthenticated subscribers with low computational overhead and without violating the decoupling, the privacy-preserving Pub/Sub design of Galaxy includes two phases: secret key sharing and encrypted Pub/Sub. In the secret key sharing phase, different organizations use threshold encryption together with an access control list to share the symmetric keys used for SSE and AES through the governance ledger without destroying the decoupling nature of the system. In the encrypted Pub/Sub phase, Galaxy adopts symmetric searchable encryption to encrypt the publication headers and subscriptions for secret matching and uses AES to encrypt the publication payload. We introduce more details of our privacy-preserving Pub/Sub design in Section V.

IV. BFT Pub/Sub Design

In this section, we elaborate on how Galaxy achieves BFT Pub/Sub. First, we present how Galaxy assigns brokers into different shards in a partially random manner. Second, we list the client APIs related to the BFT Pub/Sub service and their workflows provided by brokers within each shard. Finally, a novel leader rotation mechanism is proposed to improve the performance of the leader-based BFT consensus.

A. Shard Assignment

Sharding is one of the most commonly used horizontal scaling methods in distributed systems. Unlike traditional distributed systems, since malicious nodes need to be less than a certain proportion, the use of sharding in blockchain systems requires a trade-off between safety and scalability. Being a blockchain-based Pub/Sub system, Galaxy has to

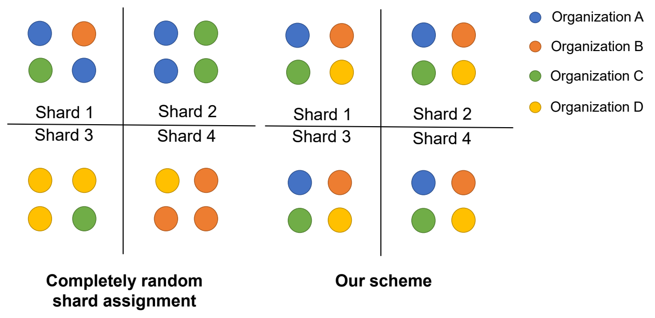


Fig. 2: An example of comparison between completely random shard assignment and our scheme.

ponder over how to assign nodes to shards in the data layer to limit the proportion of malicious nodes within each shard. Most of the existing blockchain systems employ a completely random assignment method to achieve probabilistic security [46], [47]. However, we argue that a completely random method does not conform to the real requirements of the permissioned blockchains and the proportion of nodes between organizations within each shard should be taken into account. This idea comes from a key observation: In permissioned blockchains, the units of trust are organizations rather than independent nodes. While organizations usually have enough computing resources to deploy a large number of nodes to meet the probabilistic security within each shard, the number of organizations is often limited. All the nodes within an organization may intentionally collude and break the safety of BFT consensus, so it is necessary to control the proportion of nodes belonging to each organization within each shard. A simple example of completely random shard assignment is shown on the left of Fig. 2. In this case, the number of nodes belonging to an organization within each shard is likely to exceed one-third, so a malicious organization can easily destroy the safety of a shard, such as the Organization B within the Shard 4. More formally, we can use the hypergeometric distribution to estimate the probability of selecting a faulty shard. We assume the overall network size $K = M * N$, where M is the number of organizations and N is the number of nodes within each organization (here we assume the number of nodes in organizations is equal for simplicity). We also assume there are F out of M organizations are Byzantine. Let X be the random variable that represents the number of Byzantine nodes assigned to a shard, the probability of selecting a faulty shard of size n is:

$$P(X \geq f) = \sum_{x=f}^{n} \frac{\binom{FN}{x}\binom{K-FN}{n-x}}{\binom{K}{n}}$$

Let $f = \lfloor \frac{n}{3} \rfloor$ and $F = \lfloor \frac{M}{4} \rfloor$, each shard should contain at least 297, 492, 579, 621 nodes to keep the faulty probability under 2^{-20} when the overall network contains 2, 4, 8, 16 shards, respectively.

In this section, we propose a partially random shard assignment method so that the proportion of nodes belonging

Algorithm 1 Partially Random Shard Assignment

```
Input:
  The $M$ organizations: $O = \{O_1, O_2, ..., O_M\}$
  The $N_i$ nodes within $O_i$: $P_i = \left\{P_i^1, P_i^2, ..., P_i^{N_i}\right\}$, $1 \leq i \leq M$
Output:
  The $S$ shards: $D = \{D_1, D_2, ..., D_k\}$

1: Initialize $D_k \leftarrow \emptyset$ for each $1 \leq k \leq S$
2: for each organization $O_i$ in $O$ do
3:     $R_i \leftarrow VRF(SK_i, O_i)$
4:     $Assigned_i \leftarrow \emptyset$
5: end for
6: for each shard $D_k$ in $D$ do
7:     $C \leftarrow \lceil \frac{\min N_i}{S} \rceil, \ 1 \leq i \leq M$
8:     for each organization $O_i$ in $O$ do
9:         while $|Assigned_i| < C$ do
10:            Randomly select $P_i^j$ based on $R_i$
11:            if $P_i^j \notin Assigned_i \vee |Assigned_i| = N_i$ then
12:                $D_k \leftarrow D_k \cup P_i^j$
13:                $Assigned_i \leftarrow Assigned_i \cup P_i^j$
14:            end if
15:        end while
16:    end for
17: end for
18: return $D$
```

to each organization in each shard is equal, as depicted on the right of Fig. 2. In our scheme, as long as the number of Byzantine organizations participating in the system is less than one-third, the shard assignment can achieve deterministic security regardless of the shard size. The shards assignment process is given in Algorithm 1. We assume that the number of shards is S, and D_k represents the kth shard. All organizations participating in the system register their brokers in the governance ledger. We assume there are M organizations ($M \geq 4$) and each organization O_i registers N_i nodes. The id of each node is P_i^j, where i represents the organization it belongs to and j represents its number within the organization. Each organization also generates a random number R_i and a proof $\pi(R_i)$ by using a verifiable random function (VRF) [48] based on its public key SK_i:

$$R_i \leftarrow VRF(SK_i, O_i)$$
$$\pi(R_i) \leftarrow VRF\_Proof(SK_i, O_i)$$

Here R_i is used as a random source for the node assignment process within each organization and $\pi(R_i)$ can be used to verify the authenticity of R_i by other organizations with the public key PK_i of O_i. For each shard D_k, Galaxy randomly selects C nodes within each organization O_i by computing a permutation of the nodes identifiers based on R_i. The selection of C here needs to ensure that the nodes in each shard can be evenly selected from all organizations, even for the organization with the least number of nodes:

$$C = \lceil \frac{\min N_i}{S} \rceil, \ 1 \leq i \leq M$$

In other words, Galaxy uniformly selects a total of $M\lceil \frac{\min N_i}{S} \rceil$ nodes from all the organizations for each shard. We choose $\lceil \frac{\min N_i}{S} \rceil$ as the number of nodes selected in each organization because we try to avoid the situation where one node is assigned to different shards, while ensuring that each organization has a sufficient number of nodes to be assigned. In addition, Galaxy gives priority to selecting nodes that have not been assigned to a shard. However, when the number of nodes is insufficient, some nodes may need to participate in more than one shard, which will bring greater computation and storage overhead to such nodes, and thus there is a trade-off between performance and safety. Moreover, for organizations with an insufficient number of nodes, they can choose to register more nodes in the next shards assignment phase; when the number of registered nodes in the organization is too large, some nodes within the organization will not be allowed to join any shards and the organization can choose to run CFT consensus algorithms between nodes within its organization to achieve higher availability.

After the shard assignment is completed, each shard runs its BFT consensus instance independently, providing Pub/Sub services for a range of topics. To determine the corresponding shard for each topic, our current implementation maps each topic to the target shard D_{topic} based on hashing:

$$D_{topic} = Hash(topic) \ mod \ S + 1$$

For resource constrained IoT clients, they can choose to rely on trusted proxies to forward messages to the corresponding shard. Notice that we choose this load balancing strategy only for the implementation simplicity and other application-specific load balancing methods can also be used for Galaxy, which is orthogonal to our work.

B. Pub/Sub Workflow

In each shard of the data layer, Galaxy provides a list of APIs for clients to perform BFT Pub/Sub. We assume that the message sent by the client can be expressed as $\langle op, data \rangle$, where op represents the operation type, and $data$ is the data conforming to a specific operation type. In this section, we describe the client APIs provided by Galaxy and their workflows in detail:

Authenticate: First, each client pre-registers an identity and obtains a verifiable token using the TA within its organization. The client sends the token in the $data$ field to a broker within the same organization, and the broker decides whether to establish a connection with the client after verifying the token. We did not choose to achieve consensus on the client's authentication message among brokers for two reasons. First, the malicious behaviors of the clients, such as spam attacks, are not the main concern in the permissioned blockchain [49]. Second, the client's signature is included in subsequent messages and recorded on the blockchain, enabling subsequent auditing.

Subscribe: The client sends subscriptions including a list of topics in the $data$ field to the broker, and the broker broadcasts it to all brokers. Next, the BFT consensus algorithm runs among the brokers to reach an agreement on the subscription.

Here we choose to implement HotStuff [45] as the consensus module of the broker to ensure performance and responsiveness. We briefly introduce a generic phase in HotStuff: The leader of the current view batches messages into a block and broadcast it to all brokers. Brokers verify the block and send a vote to the leader of the next view, and the next leader will collect a quorum of votes to form a quorum certificate (QC) and include the QC in the next proposed block. When a block is certified by three QCs, the block is considered committed. Our implementation is based on the event-driven HotStuff and we refer the readers to the raw paper for full details [45]. When the subscription is committed, the consensus module hands over the subscription to the Pub/Sub module, and the Pub/Sub module stores the subscription in the local key-value database. Finally, the broker to which the client is connected sends a reply to the client indicating the successful commitment of the subscription.

It is worth noting that Galaxy does not force the client to wait for at least $f+1$ replies to ensure safety like traditional BFT SMR. In Galaxy, clients can choose which trusted brokers to connect to and use the replies for verification based on their own conditions. Here the number of brokers, which is called a custom quorum by us, is a tunable parameter in our system. We made this design choice for three reasons: First, in our system model, we assume clients only establish connections with brokers belonging to their own organization, so there is a trust relationship between the clients and the brokers. The way of obtaining information from some trusted nodes is common even in permissionless blockchains. Second, in order to receive at least $f+1$ replies, the client needs to maintain connections with all or at least most of the brokers, which is impractical for some resource-constrained clients in IoT. Finally, the user can broadcast a Read operation, which we will describe below, to collect at least $f+1$ replies to ensure the safety of particular messages. Galaxy also uses a similar way to send replies to the clients who initiate the *Unsubscribe* and *Publish* operations.

Unsubscribe: Similar to the Subscription operation, the client sends a list of topics to the brokers to perform unsubscription. After the consensus between the brokers is completed, the Pub/Sub module of each broker is responsible for deleting the corresponding subscriptions in the local storage. The broker connected by the client sends a reply to the client indicating the success of the *Unsubscribe* operation.

Publish: The client sends a publication in the $data$ field to the broker. Each publication contains a *header* and a *payload*. The header contains a series of topics and other metadata. Similar to the subscription operation, the publication is first replicated among brokers through the consensus module. After the consensus, the Pub/Sub module of each broker will check whether there is a subscription from the clients currently connected that matches the topics in the publication header. If yes, the broker will forward the publication to the corresponding client. Otherwise, the publication is discarded by the Pub/Sub module. The broker connected by the client also sends a reply to the client indicating the result of the *Publish* operation.

Read: Although Galaxy is a push-based Pub/Sub system, we also provide a simple *Read* API for clients to obtain messages for verification proactively. The client broadcasts a *Read* operation to all or a part of brokers to request a specific block by specifying the block number in the $data$ field. These brokers return the corresponding block to the client. The user needs to collect at least $f+1$ replies to ensure absolute safety. However, since the brokers discard the blocks that have not been subscribed to and do not record the locations the clients have read, the *Read* operation in Galaxy is only an auxiliary operation and does not provide the same guarantee as those pull-based Pub/Sub systems [7], [10], [13].

CSTO Publish: While intra-shard total order may be adequate in most situations, sometimes clients want to realize the total order among publications across different shards [10], [50]. In particular, Galaxy's governance ledger determines a global total order of the cross-shards publications by implementing an optional *Cross-Shards Total Order (CSTO) Publish* API, which is used to verify and order publications from different shards.

To determine the cross-shard total order of a publication, the sender of the publication changes the op tag in the publication from *Publish* to *CSTO Publish*. Here we denote a CSTO publication as P. After the consensus on the block B containing P is reached, the broker, to which the sender connected, sends a transaction in the form of $\langle op, H_p, H_{root}, Proof, QC, desc\rangle$ to the governance ledger, where op represents the transaction type (*CSTO Publish*), H_P is the hash of P, H_{root} represents the Merkel tree root hash of the block B, $Proof$ is the Merkle proof proving that P belongs to the Merkle tree with root H_{root}, QC is a list of quorum certificates (in HotStuff, QC needs to form a three-chain) showing that the block with H_{root} has been committed, and $desc$ is a description of the publication to help the governance ledger perform ordering. The governance ledger runs another BFT consensus to verify and order this CSTO transaction. The ordering can be performed based on any deterministic methods, such as shard id, timestamp, or other specific application semantics through $desc$. After a list of CSTO publications are committed in the governance ledger, clients can determine the cross-shard total order of these publications by querying the governance ledger.

C. CWA Leader Rotation

In the leader-based BFT consensus like HotStuff, the leader rotation usually adopts a round-robin scheme. Although this scheme is simple and fair, it inevitably elects malicious nodes into the leader periodically, which can significantly degrades the performance of the consensus algorithms such as HotStuff which rotates the leader frequently. Due to the complexity of the Byzantine behaviors, we focus on mitigating the following two problems:

Crash: *Crash* here means that the node appears to be suspended, that is, the faulty node does not propose messages or participate in consensus voting. This problem can come from the failure of honest nodes or the intentional behavior of malicious nodes, which is indistinguishable in the Byzantine model. Galaxy chooses to consider this problem because it is an extremely common fault in a large-scale distributed system where different servers may go down frequently.

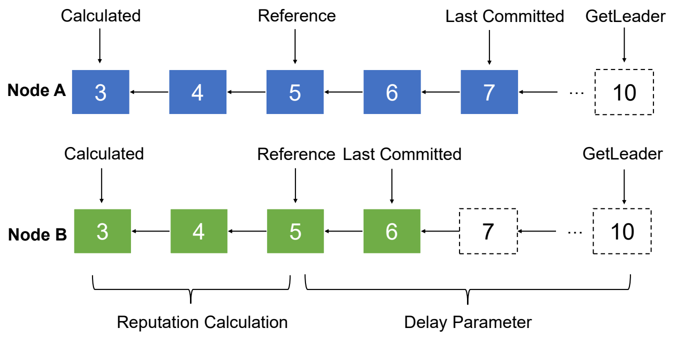


Fig. 3: An simple example to show the basic idea of the CWA algorithm. Dashed blocks represent blocks that have not yet been committed and the number on each block represents its view number. Node A and B both want to get the leader of view 10, but their underlying blockchains are different (node A has committed the blocks to view 7 but node B has only committed the blocks to view 6). By specifying a delay parameter, they locate the same reference block (view 5) committed by both nodes and use the same prefix of the blockchain to calculate the reputation.

Algorithm 2 CWA Leader Rotation Algorithm

```
// Return the leader of view $V$
function GetLeader($V$)
    // Get the last committed block and its view number
    $B_{lc} \leftarrow$ GetLastCommittedBlock(V)
    $V_{lc} \leftarrow B_{lc}$.View( )

    // Get the reference view
    $V_{ref} \leftarrow V - D$
    if $V_{ref} > V_{lc}$ then
        // Fallback to round-robin
        return RoundRobin($V$)
    end if

    // Get the reference block by its view number
    $B_{ref} \leftarrow$ GetBlockByView($V_{ref}$)

    // Update the reputation map
    for $V_i \leftarrow V_{cal},\ V_{ref}$ do
        $B_i \leftarrow$ GetBlockByView($V_i$)
        $M[B_i.\text{Proposer}(\,)] \leftarrow M[B_i.\text{Proposer}(\,)] + R_p$
        for $P_{voter}$ in $B_i$.Voters( ) do
            $M[P_{voter}] \leftarrow M[P_{voter}] + R_v$
        end for
        for $j \leftarrow 1,\ N$ do
            $M[P_j] \leftarrow \alpha \cdot M[P_j]$
            if $M[P_j] > R_{max}$ then
                $M[P_j] \leftarrow R_{max}$
            end if
        end for
    end for
    $V_{cal} \leftarrow V_{ref}$

    // Get voters of the reference block
    $Voters_{ref} \leftarrow B_{ref}$.Voters( )
    // Find the first f proposers of the reference block
    $P_{exclude} \leftarrow \emptyset$
    while $|P_{exclude}| < f \ \wedge\ B_{ref} \neq B_{genesis}$ do
        $P_{exclude} \leftarrow P_{exclude} \cup B_{ref}$.Proposer( )
        $B_{ref} \leftarrow B_{ref}$.Parent( )
    end while

    // Choose the candidates of the leader
    $Candidates \leftarrow Voters_{ref} \setminus P_{exclude}$

    // Eelect the leader
    $Leader \leftarrow \underset{P \in Candidates}{\arg\max} \ M[P]$

    return $Leader$
end function
```

Withholding: Here we use *Withholding* to refer to the type of malicious node that does not propose messages when acting as a leader, but participates in voting normally. This is a simple but powerful attack strategy in a partial-synchronous consensus like HotStuff: the system needs to wait for Δ time to trigger a timeout and elect the next leader, and usually the implementation will extend the parameter Δ after each timeout to ensure the liveness. As a result, frequently electing a *Withholding* node as the leader can lead to a long-term loss of liveness of the system. To make matters worse, *Withholding* nodes are hard to blame in this case, since they can attribute their behaviour to a temporary network asynchrony.

A recent work [15] selects the next leader from the voters of the latest committed block to avoid the crash nodes. However, their method lacks specifications on the deterministic approach used to select the leader, which may hinder honest nodes from reaching a consensus on the latest committed block due to network asynchrony. Inspired by their work, we designed a *crash and withholding avoidance (CWA)* leader rotation algorithm to elect the leader in each view. In particular, our approach addresses the following two challenges: First, consensus needs a leader, and a leader election also needs to reach a ”consensus” among all honest nodes. In order to avoid falling into a chicken-and-egg problem, in the CWA algorithm, each node calculates a reputation map based on its local blockchain. Honest nodes can get the same reputation map, and use a deterministic method to select the same node with a high reputation. We also added an attenuation and an upper limit of the reputation of each node to avoid monopoly. Secondly, due to network asynchrony, the last block committed by honest nodes may be different, that is, the blockchain of an honest node may be a prefix of the other, which can cause the honest nodes to fail to agree on the same reputation map for a long time. To address this challenge, we introduce the concept of *reference block* instead of relying on the last committed block to ensure that honest nodes consider the same blockchain prefix and get the same reputation map. Additionally, we also define a tunable delay parameter to trade off between capturing the recent behavior of nodes and possible temporary inconsistency. Moreover, we use the round-robin algorithm as a fallback to ensure liveness in the case of occasional temporary inconsistency when the reference block has not been committed. Fig. 3 illustrates a simple example of the CWA leader rotation algorithm to demonstrate its basic idea.

The complete process of the CWA leader rotation algorithm is given in Algorithm 2. It is a function that runs independently on each node, the input is a view number V, and the return value is the node id of the view V's leader. First, we get the view number V_{lc} of the last committed block (note that V_{lc} may be different for different honest nodes) and we calculate the reference view V_{ref} by subtracting a delay parameter D from V:

$$V_{ref} \leftarrow V - D$$

For different nodes within the same view, V and D are the same and thus their calculated reference views are also the same. Therefore, honest nodes all consider the same blockchain prefix formed by the reference view and its previous view blocks. If V_{ref} is greater than V_{lc}, that is to say, the current node has not committed the block of V_{ref}, then the node falls back to the round-robin. In this case, the nodes within the system may enter a temporary inconsistency, but they can eventually reach a consensus by using the round-robin fallback and making V_{lc} exceed V_{ref}. In addition, the likelihood of this inconsistency can be reduced by increasing D, but a larger D also means that only the behavior of earlier views of the nodes can be captured, which is a trade-off depending on the implementation and deployment environment. In addition, in HotStuff, D needs to be greater than 3 to ensure that the reference block can be committed. If V_{ref} is less than V_{lc}, the node updates its local reputation map M with blocks from view $V_{cal}+1$ to V, where V_{cal} is a local variable storing the view number of the last block that has been calculated in the reputation computation, the key of M is the id of each node and the value is its reputation. CWA algorithm adds R_p to the reputation of the proposer of each block, and R_v to the reputation of each voter:

$$M[B_i.\text{PROPOSER}(\,)] \leftarrow M[B_i.\text{PROPOSER}()] + R_p$$
$$M[P_{voter}] \leftarrow M[P_{voter}] + R_v$$

In order to avoid monopoly, Galaxy also multiplies the reputation value of each node by a decay parameter $\alpha\,(\alpha < 1)$ for each block, and sets a reputation upper limit R_{max}. After updating the reputation map, CWA algorithm chooses leader candidates in a similar way to [15] to ensure chain-quality, that is, we exclude the proposers set $P_{exclude}$ of the first f blocks from the voter of the reference block:

$$Candidates \leftarrow Voters_{ref} \setminus P_{exclude}$$

Finally, CWA algorithm selects the node with the highest reputation from the candidates set as the leader of view V.

V. Privacy-Preserving Pub/Sub Design

In this section, we introduce the Pub/Sub privacy protection scheme in Galaxy. We first analyze Pub/Sub privacy requirements and challenges of a Pub/Sub system and give an overview of our scheme by introducing the design philosophy of Galaxy, and then we describe the two phases of our proposed scheme in detail: the secret key sharing phase and the encrypted Pub/Sub phase.

A. Design Philosophy

Although the BFT consensus allows nodes to ensure data consistency even in the presence of malicious nodes, one problem remains unsolved: there is a large amount of privacy-sensitive data in the IoT, such as medical records, geographic locations, etc. In the absence of effective privacy-preserving methods, passive attackers can easily obtain the content of clients' subscriptions and publications, and the transparency of the blockchain makes the situation even worse. In reality, the senders may only want to share information among the users they designate. Therefore, it is necessary to design a scheme to achieve fine-grained privacy protection of messages without destroying the advantages brought by the blockchain. In particular, Galaxy aims to achieve the following two requirements:

- **Invisibility to Unauthenticated Subscribers:** If a correct publisher submits a publication P on a topic T, then P should not be accessed by subscribers who do not subscribe to the topic T or do not have access right on P.
- **Invisibility to Brokers:** Brokers need to be able to match publications and subscriptions without knowing their actual content.

However, it is not easy to meet the above requirements under a Pub/Sub system. Specifically, Galaxy addresses the following two prominent challenges:

- **Achieve Low Computational Overhead:** ABE is a promising encryption method that implements fine-grained access control and is adopted in several existing works. However, for high-frequency Pub/Sub operations, using asymmetric encryption like ABE will bring huge computation overhead. In Galaxy, we choose to use symmetric encryption to reduce computation overhead. In particular, we apply dynamic SSE to subscriptions and the headers of publications to implement secret matching between subscriptions and publications.
- **Guarantee the Inherent Decoupling of Pub/Sub:** a direct key exchange between publishers and subscribers would break the decoupling of the system while relying on a trusted third party could be a single point of failure. In Galaxy, we choose to apply threshold encryption to encrypt and share symmetric secret keys in a decentralized manner with the help of the governance ledger. Our scheme ensures that the key requester can decrypt the symmetric key used by the Pub/Sub encryption only after obtaining the decryption shares from at least $f+1$ nodes (i.e. at least one honest node).

In general, Galaxy's privacy-preserving Pub/Sub scheme is divided into two phases: the key sharing phase and the encrypted Pub/Sub phase. Next, we will elaborate on these two phases respectively.

B. Secret Key Sharing

In this section, we describe how Galaxy uses a tag-based threshold encryption scheme to share the symmetric secret keys among blockchain nodes in the Pub/Sub process. Our threshold encryption scheme extends the Ghadafi's distributed tag-based encryption scheme [51] by deploying Shamir secret sharing [52] to compute the secret keys of nodes.

Setup: We assume that the number of nodes in the governance ledger is n, and we set up an $(f+1, n)$ threshold encryption scheme. A threshold encryption *Setup* algorithm takes the total number of nodes n, the threshold parameter t $(t = f+1)$ and a security parameter λ as inputs, and output (TPK, TSK, TVK), where TPK is a public key, $TSK = (TSK_1, TSK_2, \ldots, TSK_n)$ is a list of private keys, and $TVK = (T = (TSK_1, TVK_2, \ldots, TVK_n)$ is a list of verification keys. TPK is used as a systems parameter in Galaxy and each node P_i of the governance ledger has its own private key TSK_i and TVK_i, $1 \leq i \leq n$. In addition, a client A generates its SSE secret key pair (SK_{sse}, SK_{aes}) by using the TA in its organization, where SK_{sse} is the secret key used by SSE and SK_{aes} is the secret key used by AES.

Key Encryption: The client A runs a threshold encryption algorithm *Encrypt*, takes the secret key pair (SK_{sse}, SK_{aes}), an access control list ACL, and the threshold public keys TPK as inputs, and outputs the encrypted key pair C_{key}. Here ACL is used as the tag to prevent chosen ciphertext attack.

Key Publication: To make the secret key pair available to other clients, the client A attaches the access control list ACL to C_{key} (here our implementation is based on organization id access control, but it can also be extended to attribute-based or other access control), and sends a key publication transaction including (ACL, C_{key}) to the governance ledger. During the consensus, each node P_i runs a ciphertext validation algorithm $IsValid$ to verify the transaction. $IsValid$ takes the public key TPK, the access control list ACL, and the encrypted key pair C_{key} as inputs, and outputs 1 if C_{key} is valid or 0 other wise. After the consensus, (ACL, C_{key}) is replicated at every node in the governance ledger if if the transaction is valid .

Decryption Share Generation: A client B tries to get the latest key published by a client A from the governance ledger by broadcasting a request to all the nodes in the governance ledger. Each node in the governance ledger independently verifies whether the organization to which client B belongs satisfies the ACL issued by client A. If yes, the node i runs an threshold decryption algorithm $Decrypt$ that takes the encrypted key pair C_{key} and its threshold private key TSK_i as inputs, and outputs a decryption share σ_i. Each node Pi sends its decryption share σ_i with its verification key TPK_i to client B respectively. Since our system model assumes that the attacker cannot compromise the communication channel between the honest nodes and clients, the attacker cannot obtain more than f decryption shares and thus the original message cannot be recovered.

Key Recovery: After receiving each decryption share, the client B can run a share verification algorithm $ShareVerify$ which takes the public key TPK, the verification key TPK_i, the access control list ACL, the encrypted key pair C_{key}, and the decryption share σ_i as inputs, and outputs 1 if σ_i is valid or 0 otherwise. After collecting at least $f+1$ valid decryption shares, the client B runs a message recovery algorithm *Combine*. The algorithm takes the encrypted key pair C_{key} and the $f+1$ decryption shares $(\sigma_1, \sigma_2, \ldots, \sigma_{f+1})$ as input, and outputs the raw secret keys (SK_{sse}, SK_{aes}) used in the encrypted Pub/Sub phase, which we will introduce next.

C. Encrypted Pub/Sub

In this section, we introduce how to encrypt publications and subscriptions based on the keys published on the governance ledger and how the brokers complete anonymous matching of publications and subscriptions based on the dynamic SSE scheme [20].

Subscribe: We define a subscription as S and the list of topics included in S as T. A subscriber runs a *Setup* algorithm, takes the SSE secret key SK_{sse} and the topic list T as inputs, and outputs an encrypted topic list ET. The subscriber replaces T in the subscription S with ET and sends S to the brokers. ET is stored by the brokers for later searching.

In order to add new topics to an existing subscription, the subscriber runs an *InsertToken* algorithm that takes SK_{sse}, the list of new topics to be added T_{new} as inputs, and outputs an insert token itk. The subscriber replaces T in the subscription S with itk and an *add* operator, and sends S to the brokers using the *Subscribe* API. Each broker processes the subscription S after consensus and runs an *Insert* algorithm which takes ET and itk as input, and generates a new topic list ET'.

Unsubscribe: To delete existing topics from an existing subscription, the subscriber runs an *DeleteToken* algorithm that takes SK_{sse}, the list of existing topics to be deleted T_{del} as inputs, and outputs a delete token dtk. The subscriber replaces T in the unsubscription US with dtk and a *delete* operator and sends US to the brokers using the *Unsubscribe* API. Each broker processes the unsubscription US after consensus and runs an *Delete* algorithm which takes ET and dtk as input, and generates a new topic list ET'.

Publish: We define a publication as P. For the topic list T_{pub} in the header of P, a publisher runs a *SearchToken* algorithm to encrypt the topic list as a SSE search token. It uses the SSE secret key SK_{sse} and T_{pub} as inputs and outputs a search token stk. For the payload, P_{pub} of P, the publisher uses the AES secret key SK_{aes} to encrypt P_{pub} as a ciphertext C_{pub}. The publisher sends the encrypted publication P including stk and C_{pub} to the brokers. After the consensus, each broker runs a *Search* algorithm. It takes the stk in the header of P, the SSE secret key SK_{sse} and the encrypted database containing all subscriptions EDB as inputs and outputs all the subscription identifiers U_{match} matching the topic list T_{pub}. For the subscriber Sub_i of each subscription S_i in U_{match}, if Sub_i is connected to the current broker, then the broker forwards P to Sub_i. After receiving P, Sub_i runs an AES decryption algorithm. The decryption algorithm uses SK_{aes} and the encrypted payload C_{pub} as inputs and outputs the origin publication P_{pub}.

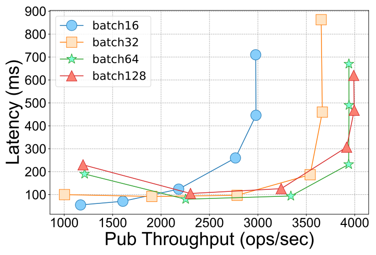


Fig. 4: Throughput and latency of Pub in the LAN setting

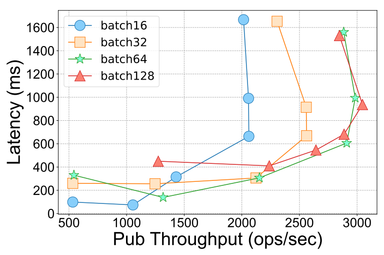


Fig. 5: Throughput and latency of Pub in the WAN setting

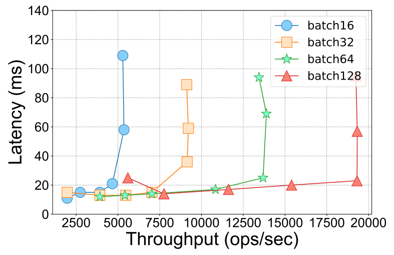


Fig. 6: Throughput and latency of CSTO transactions

VI. Evaluation

In this section, we first introduce our implementation and experimental setup, and then we present the evaluation results as follows:

1) The overall performance of Galaxy under different experimental settings.
2) The BFT scalability and resilience of Galaxy under different experimental settings.
3) The time costs of cryptographic primitives used in Galaxy and its comparison with Cui et al's scheme [23].

A. *Implementation and Evaluation Setup*

We implemented a prototype of Galaxy in Go. The network module is implemented with a gRPC wrapper Gorums [53] to invoke quorum calls. The consensus module is implemented based on the event-driven HotStuff, and interacts with other modules by providing interfaces such as message proposal, verification and commitment. The storage module maintains a blockchain and an in-memory key-value database. The blockchain is an append-only log which records all the committed messages. The key-value database records the current user subscriptions to facilitate the query of the broker, similar to the world state in the Hyperledger Fabric [33]. In order to reduce the storage overhead, a broker can choose to periodically prune the unwanted part or use checkpoint technology in the append-only ledger. The crypto module implements cryptographic primitives used by the consensus module and the Pub/Sub module. We implement the adopted threshold encryption scheme in Go and implements the bilinear group operations with the bn256 package [54]. For the SSE scheme, we used a dual secure SSE scheme [20] by extending and wrapping the Clusion Java library [55].

We deployed the prototype on 16 Alibaba Cloud ecs.c7.xlarge servers (each with 4 vCPU and 8GB memory). We assume that different servers belong to different organizations. In different experimental settings, a controller that implements our shard assignment strategy evenly distributes brokers across different shards.

B. *Overall Performance*

Performance of the data layer: First, to demonstrate the performance of the data layer, we deployed a total of **128** brokers (8 brokers on each server) and **1280** clients (10 clients connecting to each broker). Brokers are divided into 4 shards (32 brokers per shard) and different shards run in parallel. We chose to take the throughput and latency of the **publish (with SSE)** operations as the performance metric and we do it for two reasons: First, publish operations are more frequent than subscribe operations in most scenarios. Second, publish operations (with SSE) are more expensive than publish (without SSE) and subscribe operations since brokers need to invoke the SSE search primitive. As a result, using publish (with SSE) as the performance metric can show the upper limit of the performance of the system. We also emphasize that Galaxy can achieve better performance when only part of the messages in the system requires privacy protection. In the following parts, we abbreviate the publish (with SSE) operation as Pub operation.

We set the size of each pub operation to **128** bytes (in the follow-up experiments, unless otherwise specified, we keep the size of a single operation at 128 bytes). We conduct the evaluation under two different network settings: **LAN** (3 Gbps bandwidth per server) and **WAN** (50Mbps bandwidth per server). We also adjusted different batch sizes (16, 32, 64, 128) under each network setting, and the experimental results are shown in Fig. 4 and Fig. 5. The results show that when batch size = 128, the maximum throughput reaches **3994 ops/sec** and **3047 ops/sec** in LAN and WAN settings, respectively. The peak throughput is lower and the average latency is higher in the WAN, which is reasonable due to the constrained bandwidth. In addition, the peak throughput generally increases with the increase in batch size. However, when the batch size reaches 128, the maximum throughput no longer increases significantly and even decreases. This is because the time spent waiting for a batch of messages gradually exceeds the time cost of the consensus.

Performance of the governance layer: The performance of the governance ledger is also tested. Since we assume the governance ledger is a small-scale permissioned blockchain, 16 nodes (1 node on each server) and 80 clients (5 clients for each node) are deployed and the **LAN** network setting is adopted. We choose the throughput and latency of the **CSTO transaction** as the performance metrics. Fig. 7 shows the performance of the governance ledger under different batch sizes. The peak throughput of the governance leader reaches **19,323 ops/secs** when batch size = 128, which is significantly increased compared with the brokers at the data

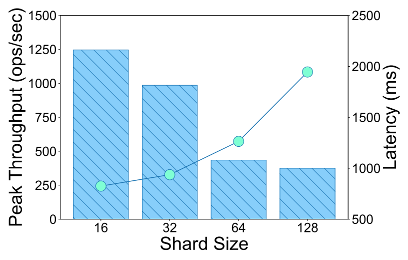


Fig. 7: Peak throughput and corresponding latency of Pub under different shard sizes

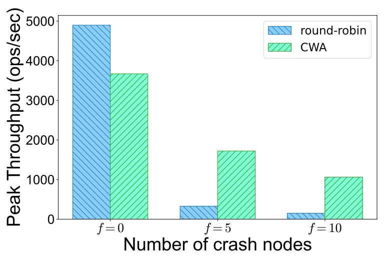


Fig. 8: Peak throughput of a single shard (with 32 brokers) under different numbers of crash nodes

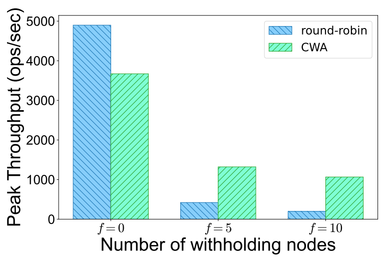


Fig. 9: Peak throughput of a single shard (with 32 brokers) under different numbers of withholding nodes

layer. In addition to having higher computing and bandwidth resources, the implementation of governance leader removes the Pub/Sub-related modules and only adds a lightweight smart contract module, so the overhead is greatly reduced compared with the broker shards at the data layer.

C. BFT Scalability and Resilience

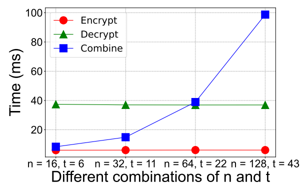
Scalability: To get a view of the scalability of Galaxy, we also evaluated the performance of a single shard with different numbers of brokers. To ensure that the computing resource of each node is equal, we still deploy 8 brokers on each server and gradually increase the number of servers from **2** to **16**. We use the **WAN** network setting in this experiment. Fig. 6 shows the peak throughput and corresponding latency of a single shard under different shard sizes. The result shows that peak throughput decreases with the increase of the shard size, which is reasonable because of the higher communication and cryptography overhead. We also notice that the performance decreases significantly when the shard size increases from 32 to 64 and the reason is that the bandwidth and the limited computation resources of the servers have reached the bottleneck at this time. We suggest selecting the largest partition size when the bottleneck is reached in the actual deployment to increase the security of a single partition. In addition, since all the shards in Glaxay run in parallel (except for possible CSTO transactions), large-scale horizontal expansion can be carried out by increasing the number of partitions when the computing resources are sufficient and the governance ledger does not reach the performance bottleneck.

Resilience: In this part, we show the resilience of the system by comparing the peak throughput of the Pub operation under the round-robin and the proposed CWA leader rotation algorithm respectively. The experiment is carried out in a single shard of **32** brokers (2 brokers on each server) under the **WAN** setting. For the CWA algorithm, we set the proposer reputation R_p = 10, the voter reputation R_v = 1, the delay parameter D = 5, the attenuation parameter α = 0.8, and the reputation upper limit R_{max} = 100. We simulated the two types of malicious nodes respectively: **crash** node and **withholding** node. Fig. 8 and Fig. 9 show the peak throughput of the system when the number of these two kinds of malicious nodes (denoted as f) increases. When f = 0, the peak throughput of the system under the CWA algorithm decreases compared with the round-robin because the CWA algorithm introduces more complex computation. However, when the number of malicious nodes increases, the CWA algorithm significantly outperforms the round-robin. When f = 5, the CWA algorithm achieves about **4x** and **2x** peak throughput improvement over the round-robin algorithm under the crash and withholding faults respectively. When f = 10, the CWA algorithm outperforms the round-robin by a factor of **6x** and **4x**.

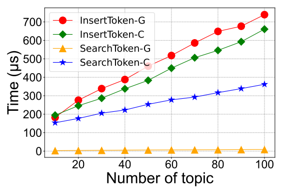
D. Privacy-Preserving Computation Cost

In this section, we demonstrate the time cost of the cryptographic primitives used by Galaxy. For each cryptographic primitive, we pick the average time consumption of 100 experiments. For the secret key sharing phase, we demonstrate the time cost of different operations under different combinations of n and t. The secret keys to be shared are two 32 bytes random string used for SSE and AES. For the encrypted Pub/Sub phase, we compare our scheme (denoted by **Operation-G**) with Cui et al.'s scheme [23] (denoted by **Operation-C**). The privacy-preserving Pub/Sub scheme proposed by Cui et al. encrypts the subscription and publish header based on an SSE scheme SUISE [56], and the publish payload is encrypted based on a lightweight KP-ABE scheme [57]. For the sake of fairness, we implement the SUISE scheme in Java and the KP-ABE scheme in Go. We adopt the cryptographic primitives and data structures provided by the default Java libraries to implement SUISE, which is consistent with the original paper [56]. For the KP-ABE scheme, we implement the pairing-based cryptographic primitives with a PBC Go Wrapper [58]. Both our scheme and Cui et al.'s scheme utilize HMAC-SHA256 and AES-CTR as basic building blocks, and each topic is a 5-byte random string.

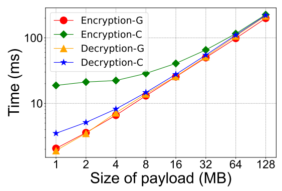
Secret Key Sharing: First, we show the time cost of threshold encryption primitives in the secret key sharing phase. Fig. 10(a) shows the time cost of *Encrypt*, *Decrypt* and *Combine* operations under different combinations of the total number of shares (denoted by n) and the threshold (denoted by t). We have considered the costs of *IsValid* and *ShareVerify* operations as part of the *Decrypt* and *Combine* operations, respectively. The evaluation result shows that the time consumption of *Combine* increases with the number of n and t and is about 98.7 ms when $n = 128$ and $t = 43$. The time costs of *Encrypt*



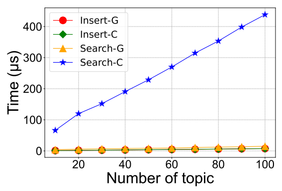
(a) The time cost of the *Encrypt*, *Decrypt* and *Combine* in the secret key sharing phase



(b) The time of the *InsertToken* and *SearchToken* primitives used to encrypt subscriptions/publication headers



(c) The time of encrypting and decrypting the publication payload



(d) The time of the *Insert* and *Search* primitives

Fig. 10: Time cost of cryptographic primitives

and *Decrypt* are about 6 ms and 37 ms, which are unrelated to n and t. This overhead is acceptable since the network size of the governance ledger is usually small and the key publication transactions are not frequent in practice. In addition, although our scheme requires the secret key sharing phase to share the symmetric keys compared with Cui et al.'s scheme, we avoid the need for the centralized TA in Cui et al.'s scheme, and the cost of our key sharing phase can be amortized during multiple Pub/Sub processes if the keys is not changed every time.

InsertToken and SearchToken Generation: *InsertToken* and *SearchToken* operations are used by clients to encrypt the subscriptions and publication headers respectively. Fig. 10(b) demonstrates the time required for generating the insert token and the search token. In this experiment, the number of topics varies from 10 to 100 (the number of topics is less than 20 in most cases according to the industry-standard benchmark [18]). Since the SSE scheme adopted by Cui et al. does not offer a special setup operation, we compared the *InsertToken* operation instead of the *Setup* operation for subscription encryption. For the generation of *InsertToken*, both our and Cui et al.'s schemes show a linear growth with the number of topics. Our scheme has lightly more expensive costs on the *InsertToken* operation because the SSE scheme adopted by Galaxy involves more complex data structures in generating *InsertToken*. However, our scheme provides a higher level of security against persistent advertisement and snapshot advertisement. The cost of generating *SearchToken* in our scheme is significantly less than that in Cui et al.'s scheme, and it is independent of the number of topics. When the number of topics is 100, the time cost of *SearchToken* in our scheme is reduced by **97.6**% compared to Cui et al.'s scheme.

Payload Encryption and Decryption: Payload *Encryption* and *Decryption* operations are used by clients to encrypt and decrypt the publication payload. Fig. 10(c) shows the encryption and decryption time of the publication payload. Both schemes show an approximate linear growth trend. In this experiment, we fix the number of attributes for the KP-ABE scheme to 20, which is the same as the setting in the Cui et al.'s paper. In fact, the AES encryption and decryption operation is part of the lightweight KP-ABE scheme adopted by Cui et al.'s scheme, so our scheme has less overhead at any time, especially when encrypting a small-size payload (the payload size is usually less than 1MB according to the experimental settings of some industrial deployed systems [7], [10]). When the payload size is 1 MB, the *Encryption* and *Decryption* in our scheme take up to 2.06ms and 1.91ms, which is reduced by **89.4**% and **45.7**% compared to Cui et al.'s scheme respectively.

Insert and Search: *Insert* and *Search* operations are used by brokers to update the encrypted subscriptions and perform encrypted Pub/Sub matching respectively. Fig. 10(d) shows the time needed for the *Insert* and *Search* operations at the brokers. For the *Search* operation, Cui et al's scheme shows a significant increase with the number of topics, because the topics in the experiment are randomly generated and the index containing previously searched words in their scheme could not be used. Other operations present almost negligible overhead independent of the number of topics. When the number of topics is 100, the *Search* operation takes up to 438.2µs in Cui et al.'s scheme while our scheme takes only 14.9µs, reducing the overhead by **96.6**%.

VII. Conclusion

In this paper, we present Galaxy, a blockchain-based Pub/Sub IoT data sharing framework. While maintaining the asynchrony, decoupling and one-to-many advantages of the Pub/Sub paradigm, Galaxy further addresses the two major challenges of applying a Pub/Sub system to the IoT: Byzantine faults and privacy. In order to address the Byzantine faults, Galaxy realizes the tradeoff between scalability and safety through a partial-random shard assignment strategy, achieves a BFT Pub/Sub workflow by introducing the streamlined BFT consensus, and adopts a CWA leader rotation algorithm to avoid frequent leader failures. In order to protect IoT data privacy, Galaxy realizes the secret sharing of symmetric keys through threshold encryption and provides efficient secret matching of publications and subscriptions based on symmetric searchable encryption. We implement a prototype of Galaxy and conduct extensive evaluation from various dimensions. The results demonstrate that Galaxy exhibits good performance and scalability in different experimental settings while also boasting reduced cryptographic computational overhead in comparison to a representative relative work.

References

[1] J. Cheng, W. Chen, F. Tao, and C.-L. Lin, "Industrial iot in 5g environment towards smart manufacturing," *Journal of Industrial Information Integration*, vol. 10, pp. 10–19, 2018.

[2] M. N. Bhuiyan, M. M. Rahman, M. M. Billah, and D. Saha, "Internet of things (iot): a review of its enabling technologies in healthcare applications, standards protocols, security, and market opportunities," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 474–10 498, 2021.

[3] F. Al-Turjman and M. Abujubbeh, "Iot-enabled smart grid via sm: An overview," *Future Generation Computer Systems*, vol. 96, pp. 579–590, 2019.

[4] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-s—a publish/subscribe protocol for wireless sensor networks," in *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*. IEEE, 2008, pp. 791–798.

[5] D. Happ, N. Karowski, T. Menzel, V. Handziski, and A. Wolisz, "Meeting iot platform requirements with open pub/sub solutions," *Annals of Telecommunications*, vol. 72, no. 1, pp. 41–52, 2017.

[6] G. S. Ramachandran, K.-L. Wright, L. Zheng, P. Navaney, M. Naveed, B. Krishnamachari, and J. Dhaliwal, "Trinity: A byzantine fault-tolerant distributed publish-subscribe system with immutable blockchain-based persistence," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2019, pp. 227–235.

[7] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, vol. 11, 2011, pp. 1–7.

[8] Apache Pulsar. Accessed: Feb. 20, 2022. [Online]. Available: https://pulsar.apache.org/

[9] Google Cloud Pub/Sub. Accessed: May 10, 2022. [Online]. Available: https://cloud.google.com/pubsub

[10] C. Ding, D. Chu, E. Zhao, X. Li, L. Alvisi, and R. Van Renesse, "Scalog: Seamless reconfiguration and total order in a scalable shared log," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 325–338.

[11] L. Lamport, "Paxos made simple," *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pp. 51–58, 2001.

[12] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (Usenix ATC 14)*, 2014, pp. 305–319.

[13] B. Huang, R. Zhang, Z. Lu, Y. Zhang, J. Wu, L. Zhan, and P. C. Hung, "Bps: A reliable and efficient pub/sub communication model with blockchain-enhanced paradigm in multi-tenant edge cloud," *Journal of Parallel and Distributed Computing*, vol. 143, pp. 167–178, 2020.

[14] N. Zupan, K. Zhang, and H.-A. Jacobsen, "Hyperpubsub: a decentralized, permissioned, publish/subscribe service using blockchains," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos*, 2017, pp. 15–16.

[15] S. Cohen, R. Gelashvili, L. K. Kogias, Z. Li, D. Malkhi, A. Sonnino, and A. Spiegelman, "Be aware of your leaders," in *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*. Springer, 2022, pp. 279–295.

[16] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 31–42.

[17] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07)*. IEEE, 2007, pp. 321–334.

[18] M. Ion, G. Russello, and B. Crispo, "Design and implementation of a confidentiality and access control solution for publish/subscribe systems," *Computer networks*, vol. 56, no. 7, pp. 2014–2037, 2012.

[19] M. R. Asghar, A. Gehani, B. Crispo, and G. Russello, "Pidgin: Privacy-preserving interest and content sharing in opportunistic networks," in *Proceedings of the 9th ACM symposium on information, computer and communications security*, 2014, pp. 135–146.

[20] G. Amjad, S. Kamara, and T. Moataz, "Breach-resistant structured encryption," *Cryptology ePrint Archive*, 2018.

[21] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1999.

[22] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1998, pp. 1–16.

[23] S. Cui, S. Belguith, P. De Alwis, M. R. Asghar, and G. Russello, "Collusion defender: preserving subscribers' privacy in publish and subscribe systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 3, pp. 1051–1064, 2019.

[24] M. Shen, H. Liu, L. Zhu, K. Xu, H. Yu, X. Du, and M. Guizani, "Blockchain-assisted secure device authentication for cross-domain industrial iot," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 5, pp. 942–954, 2020.

[25] Y. Fan, G. Zhao, X. Lei, W. Liang, K.-C. Li, K.-K. R. Choo, and C. Zhu, "Sbbs: A secure blockchain-based scheme for iot data credibility in fog environment," *IEEE Internet of Things Journal*, vol. 8, no. 11, pp. 9268–9277, 2021.

[26] X. He, Y. Zhang, and X. Wang, "A scalable nested blockchain framework with dynamic node selection approach for iot," in *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*. IEEE, 2022, pp. 108–113.

[27] L. Zhang, F. Li, P. Wang, R. Su, and Z. Chi, "A blockchain-assisted massive iot data collection intelligent framework," *IEEE Internet of Things Journal*, 2021.

[28] 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 Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2376–2387, 2020.

[29] T. Li, H. Wang, D. He, and J. Yu, "Blockchain-based privacy-preserving and rewarding private data sharing for iot," *IEEE Internet of Things Journal*, 2022.

[30] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "{ZooKeeper}: Wait-free coordination for internet-scale systems," in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.

[31] S. Duan, C. Liu, X. Wang, Y. Wu, S. Xu, Y. Yesha, and H. Zhang, "Intrusion-tolerant and confidentiality-preserving publish/subscribe messaging," in *2020 International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2020, pp. 319–328.

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

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

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

[35] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, University of Guelph, 2016.

[36] A. Bessani, J. Sousa, and E. E. Alchieri, "State machine replication for the masses with bft-smart," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2014, pp. 355–362.

[37] C. Dong, G. Russello, and N. Dulay, "Shared and searchable encrypted data for untrusted servers," *Journal of Computer Security*, vol. 19, no. 3, pp. 367–397, 2011.

[38] K. Yang, K. Zhang, X. Jia, M. A. Hasan, and X. S. Shen, "Privacy-preserving attribute-keyword based data publish-subscribe service on cloud platforms," *Information Sciences*, vol. 387, pp. 116–131, 2017.

[39] N. Attrapadung and H. Imai, "Dual-policy attribute based encryption," in *International Conference on Applied Cryptography and Network Security*. Springer, 2009, pp. 168–185.

[40] G. D. Crescenzo, J. Burns, B. Coan, J. Schultz, J. Stanton, S. Tsang, and R. N. Wright, "Efficient and private three-party publish/subscribe," in *International Conference on Network and System Security*. Springer, 2013, pp. 278–292.

[41] W. Rao, L. Chen, and S. Tarkoma, "Toward efficient filter privacy-aware content-based pub/sub systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 11, pp. 2644–2657, 2012.

[42] S. A. Gaballah, C. Coijanovic, T. Strufe, and M. Mühlhäuser, "2pps—publish/subscribe with provable privacy," in *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2021, pp. 198–209.

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

[44] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," *Cryptology ePrint Archive*, 2016.

[45] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

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

[47] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proceedings of the 2019 international conference on management of data*, 2019, pp. 123–140.
[48] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE, 1999, pp. 120–130.
[49] L. Yang, S. J. Park, M. Alizadeh, S. Kannan, and D. Tse, "{DispersedLedger}:{High-Throughput} byzantine consensus on variable bandwidth networks," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 493–512.
[50] K. Zhang, V. Muthusamy, and H.-A. Jacobsen, "Total order in content-based publish/subscribe systems," in *2012 IEEE 32nd International Conference on Distributed Computing Systems*. IEEE, 2012, pp. 335–344.
[51] E. Ghadafi, "Efficient distributed tag-based encryption and its application to group signatures with efficient distributed traceability," in *International Conference on Cryptology and Information Security in Latin America*. Springer, 2014, pp. 327–347.
[52] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[53] T. E. Lea, L. Jehl, and H. Meling, "Towards new abstractions for implementing quorum-based systems," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2380–2385.
[54] The Go bn256 package. Accessed: June 10, 2022. [Online]. Available: https://github.com/cloudflare/bn256
[55] The Clusion Library. Accessed: June 10, 2022. [Online]. Available: https://github.com/encryptedsystems/Clusion
[56] F. Hahn and F. Kerschbaum, "Searchable encryption with secure and efficient updates," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 310–320.
[57] X. Yao, Z. Chen, and Y. Tian, "A lightweight attribute-based encryption scheme for the internet of things," *Future Generation Computer Systems*, vol. 49, pp. 104–112, 2015.
[58] The PBC Go Wrapper. Accessed: May 5, 2023. [Online]. Available: https://github.com/Nik-U/pbc

Yuchao Zhang (Member, IEEE) received the B.S. degree in computer science and technology from Jilin University in 2012 and the Ph.D. degree from the Department of Computer Science, Tsinghua University, in 2017. She is currently an Associate Professor with the Beijing University of Posts and Telecommunications, Beijing, China. Her research interests include large scale datacenter networks, blockchain, federated learning, data privacy, and edge computing. She is a member of ACM and IEEE.

Xiaotian Wang received the Bachelor's degree from Beijing University of Posts and Telecommunications. He is currently pursuing the Master's degree in software engineering at Beijing University of Posts and Telecommunications. His current research interests include blockchain scalability and consensus protocols.

Xiaofeng He received the Bachelor's degree from Beijing University of Posts and Telecommunications. Currently he is a master student in the School of Computer Science at Beijing University of Posts and Telecommunications. His research interests include fault tolerance technologies, blockchain applications and blockchain scalability.

Ning Zhang (Senior Member, IEEE) is an Associate Professor in the Department of Electrical and Computer Engineering at University of Windsor, Canada. He received the Ph.D degree in Electrical and Computer Engineering from University of Waterloo, Canada, in 2015. After that, he was a postdoc research fellow at University of Waterloo and University of Toronto, respectively. His research interests include connected vehicles, mobile edge computing, wireless networking, and machine learning. He is a Highly Cited Researcher. He serves as an Associate Editor of IEEE Internet Of Things Journal, IEEE Transactions on Cognitive Communications and Networking, and IEEE Systems Journal; and a Guest Editor of several international journals, such as IEEE Wireless Communications, IEEE Transactions on Industrial Informatics, and IEEE Transactions on Intelligent Transportation Systems. He also serves/served as a TPC chair for IEEE VTC 2021 and IEEE SAGC 2020, a general chair for IEEE SAGC 2021, a track chair for several international conferences and workshops. He received 8 Best Paper Awards from conferences and journals, such as IEEE Globecom and IEEE ICC.

Zibin Zheng (Fellow, IEEE) received the Ph.D. degree from the received the Ph.D. degree from the Chinese University of Hong Kong, in 2011. He is currently a Professor at School of Data and Computer Science with Sun Yat-sen University, China. He serves as Chairman of the Software Engineering Department. He published over 120 international journal and conference papers, including 3 ESI highlycited papers. According to Google Scholar, his papers have more than 7000 citations, with an H-index of 42. His research interests include blockchain, services computing, software engineering, and financial big data. He was a recipient of several awards, including the Top 50 Influential Papers in Blockchain of 2018, the ACM SIGSOFT Distinguished Paper Award at ICSE2010, the Best Student Paper Award at ICWS2010. He served as BlockSys'19 and CollaborateCom'16 General Co-Chair, SC2'19, ICIOT'18 and IoV'14 PC CoChair.

Ku Xu (Senior Member, IEEE) received his Ph.D. from the Department of Computer Science & Technology of Tsinghua University, Beijing, China, where he serves as a full professor. He has published more than 200 technical papers and holds 11 US patents in the research areas of next-generation Internet, blockchain systems, Internet of Things (IoT), and network security. He is a member of ACM and senior member of IEEE. He has guest-edited several special issues in IEEE and Springer Journals. He is an editor of IEEE IoT Journal. He is also the Steering Committee Chair of IEEE/ACM IWQoS.