Qanaat: A Scalable Multi-Enterprise Permissioned Blockchain System with Confidentiality Guarantees

Mohammad Javad Amiri
University of Pennsylvania
mjamiri@seas.upenn.edu

Boon Thau Loo
University of Pennsylvania
boonloo@seas.upenn.edu

Divyakant Agrawal
University of California Santa Barbara
agrawal@cs.ucsb.edu

Amr El Abbadi
University of California Santa Barbara
amr@cs.ucsb.edu

ABSTRACT

Today's large-scale data management systems need to address distributed applications' *confidentiality* and *scalability* requirements among a set of collaborative enterprises. This paper presents *Qanaat*, a scalable multi-enterprise permissioned blockchain system that guarantees the confidentiality of enterprises in collaboration workflows. Qanaat presents *data collections* that enable any subset of enterprises involved in a collaboration workflow to keep their collaboration private from other enterprises. A transaction ordering scheme is also presented to enforce only the necessary and sufficient constraints on transaction order to guarantee data consistency. Furthermore, Qanaat supports data consistency across collaboration workflows where an enterprise can participate in different collaboration workflows with different sets of enterprises. Finally, Qanaat presents a suite of consensus protocols to support intra-shard and cross-shard transactions within or across enterprises.

PVLDB Reference Format:
Mohammad Javad Amiri, Boon Thau Loo, Divyakant Agrawal, and Amr El Abbadi. Qanaat: A Scalable Multi-Enterprise Permissioned Blockchain System with Confidentiality Guarantees. PVLDB, 15(11): 2839 - 2852, 2022.
doi:10.14778/3551793.3551835

1 INTRODUCTION

Emerging multi-enterprise applications, e.g., supply chain management [55], multi-platform crowdworking [9], and healthcare [15], require extensive use of *collaboration workflows* where multiple mutually distrustful distributed enterprises collaboratively process a mix of public and private transactions. Despite years of research in distributed transaction processing, data management systems today have not yet achieved a good balance among the *confidentiality* and *scalability* requirements of these applications.

One of the main requirements of multi-enterprise applications is confidentiality with respect to *data sharing* and *data leakage*. First, while public collaboration among all enterprises is visible to everyone, specific subsets of the data may need to be shared only with specific subsets of involved enterprises. For example, in a product trading, the distributor may want to keep collaboration with a farmer and a shipper involving terms of the trades *confidential* from the wholesaler and the retailer, so as not to expose the premium they are charging [50]. Second, the enterprises must prevent malicious nodes from leaking confidential data, e.g., requests, replies, and stored data.

Multi-enterprise applications also need to scalably process a large number of transactions within or across enterprises. Although sharding has been used in single-enterprise applications to address scalability, it is challenging to use sharding in multi-enterprise applications where confidentiality of data is paramount.

The decentralized nature of blockchain and its unique features such as provenance, immutability, and tamper-resistant, make it appealing to a wide range of applications, e.g., supply chain management [31, 84], crowdsourcing [9, 48], contact tracing [70] and federated learning [71].

In recent years, various permissioned blockchain systems have been proposed to address the confidentiality and/or scalability of multi-enterprise applications. Hyperledger Fabric [12] and its variants [44, 45, 75, 78] supports multi-enterprise applications and provides scalability using channels [13] managed by subsets of enterprises. However, all transactions of a channel are sequentially ordered, resulting in reduced performance. Fabric also uses private data collections [50] to manage confidential collaboration among a subset of enterprises. However, appending a hash of all private transactions to a global ledger replicated on every enterprise increases computational overhead and hence reduces throughput. Furthermore, Fabric does not address confidential data leakage by malicious nodes.

Caper [5] supports collaborative enterprises and provides confidentiality by maintaining a partial view of the global ledger on each enterprise. Caper, however, does not support: (1) confidential collaboration among subsets of enterprises (i.e., it supports internal or public transactions), (2) data consistency across collaboration workflows that an enterprise is involved in, (3) data confidentiality in the presence of malicious nodes, and (4) multi-shard enterprises.

Scalability has also been studied in the context of applications that are used within a single enterprise, e.g., AHL [35], SharPer [8] and ResilientDB [47]. However, since these systems are restricted to single-enterprise applications, they do not address the confidentiality requirement of multi-enterprise applications.

To address the above scalability and confidentiality challenges of multi-enterprise applications, we present *Qanaat*[1], a permissioned

Proceedings of the VLDB Endowment, Vol. 15, No. 11 ISSN 2150-8097.
doi:10.14778/3551793.3551835

[1] Qanaat is a scalable underground network consisting of private channels to transport water from an aquifer to the surface (i.e., an underground aqueduct).

blockchain system that supports collaboration workflows across enterprises. In Qanaat, each enterprise partitions its data into multiple *shards* to improve scalability. Each shard's transactions are then processed by a disjoint *cluster* of nodes.

To support confidential collaboration (i.e., data sharing), in addition to public transactions among all enterprises and internal transactions within each enterprise, any subset of enterprises might process transactions *confidentially* from other enterprises. As a result, Qanaat uses a novel hierarchical data model consisting of a set of *data collections* where each transaction is executed on a data collection. The root data collection consists of the public records from executing public transactions replicated on all enterprises. Each local data collection includes the private records of a single enterprise. Intermediate data collections, which are optional and are needed in case of *confidential* collaborations among subsets of enterprises, maintain private records shared among subsets of collaborating enterprises. Qanaat replicates such shared data collections on the involved enterprises to facilitate the use of shared data by enterprises in their internal transactions.

To support this collaborative yet confidential data model, Qanaat proposes a transaction ordering scheme to preserve the data consistency of transactions. In Qanaat, an enterprise might be involved in multiple data collections. Hence, the traditional transaction ordering schemes used in fault-tolerant protocols where each cluster orders transactions on a single data store do not work. In fact, while ordering transactions, Qanaat needs to capture the state of all data collections that might affect the execution of a transaction.

Finally, to prevent confidential data leakage despite the Byzantine failure of nodes, Qanaat utilizes the privacy firewall technique [39, 86] and (1) separates ordering nodes that agree on the order of transactions from execution nodes that execute transactions and maintain the ledger, and (2) uses a *privacy firewall* between execution nodes and ordering nodes. The privacy firewall restricts communication from execution nodes in order to filter out incorrect messages, possibly including confidential data.

The main contributions of this paper are:

- An infrastructure consisting of ordering nodes, execution nodes, and a privacy firewall and a data model consisting of data collections to preserve confidentiality, followed by a transaction ordering scheme that enforces only the *necessary* and *sufficient* constraints to guarantee data consistency,
- A suite of consensus protocols to process transactions that accesses single or multiple data shards within or across enterprises in a *scalable* manner, and
- Qanaat, a scalable multi-enterprise permissioned blockchain system that guarantees confidentiality.

2 MOTIVATION

Qanaat is designed to support multi-enterprise applications such as supply chain management [55], multi-platform crowdworking [9], and healthcare [15] in a scalable and confidentiality-preserving manner. To motivate Qanaat, we consider an example application based on the COVID-19 vaccine supply chain, given that it is a demanding application. The COVID-19 outbreak has demonstrated diverse challenges that supply chains face [4]. While the vaccine supply chain seems to work fine on the surface, it suffers from several crucial challenges. First, current vaccine supply chains are subjected to different types of vulnerabilities, e.g., attacks on 44 companies involved in the COVID-19 vaccine distribution in 14 countries [69], rapidly growth of the black market for COVID-19 vaccines [81], issuing fake vaccine cards [73], and loss of 400 millions of vaccine doses [24]. Moreover, while vaccines are well distributed in most developed countries, getting them to developing countries and accounting for them remains a challenge, e.g., fake vaccines [1, 33] and falsified CoviShield vaccines [67]. Currently, the COVID-19 vaccine supply chains have been over-stretched given the global supply chain challenges and the possibility of abuse and fraud stated above. At the same time, other supply chains are now under unprecedented pressure [18, 76].

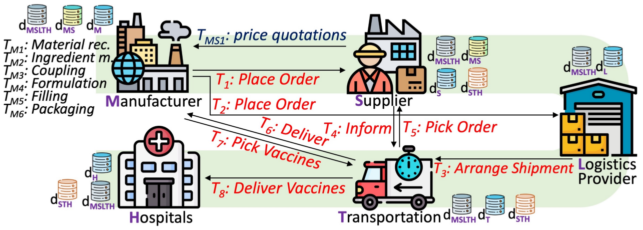

Figure 1: A vaccine supply chain collaboration workflow

Multiple enterprises collaborate in a vaccine supply chain to get vaccines from manufacturers to citizens. Enterprises involved in the vaccine ecosystem need to collaborate and share data based on agreed-upon service level agreements. Vaccines, on one hand, need to be tracked and monitored throughout the process to ensure that they are stable and not tampered with, and on the other hand, this data should be accessible to the public to increase their trust in the vaccines. While distributed databases can be used to address these challenges, they lack mechanisms to first prevent any malicious entity from altering data and second enable enterprises to verify the data and the state of collaboration. Qanaat uses blockchains to provide verifiability, provenance, transparency, and assurances to end-users of the vaccines' safety and efficacy. In particular, blockchains enable pharmaceutical manufacturers to easily track on-time shipment and delivery of vaccines, provide efficient delivery tracking for transportation companies, and help hospitals manage their stocks and mitigate supply and demand constraints.

Figure 1 shows a simplified vaccine supply chain collaboration workflow consisting of the pharmaceutical manufacturer (M), a supplier (S), a logistics provider (L), a transportation company (T), and hospitals (H). Using Qanaat, the public transactions of the collaboration workflow (T_1 to T_8) are executed on *data collection* d_{MSLTH} maintained by all enterprises.

In addition, the collaboration workflow includes the internal transactions of each enterprise. Internal transactions are executed on the enterprise's *private* data collection and might reflect patented and copyrighted formulas and information. For example, the pharmaceutical manufacturer executes its private transactions T_{M1} to T_{M6} on its private data collection d_M.

A blockchain-enabled multi-enterprise application, e.g., vaccine supply chain, needs to support the following requirements:

R1. Confidential collaborations across enterprises. Any subset of enterprises involved in a collaboration workflow might want to keep their collaboration private from other enterprises. For example, the supplier might want to make private transactions with the pharmaceutical manufacturer to keep some terms of trade confidential from other enterprises, e.g., `price quotation` (T_{MS1}) transaction. Such private transactions need to be executed on a data collection shared between *only* the involved enterprises, e.g., d_{MS} between the pharmaceutical manufacturer and the supplier.

R2. Consistency across collaboration workflows. An enterprise might be involved in multiple collaboration workflows with different sets of enterprises. For example, a transportation company that distributes both Pfizer and Moderna vaccines needs to assign trucks based on the total number of vaccine packages. In case of data dependency among transactions that span collaboration workflows, data integrity and consistency must be maintained.

R3. Confidential data leakage prevention. An enterprise needs to ensure that even if its infrastructure includes malicious nodes (i.e., an attacker manages to compromise some nodes), the malicious nodes cannot leak any confidential data, e.g., requests, replies, processed data and stored data.

R4. Scaling multi-shard enterprises. Every day, millions of people get vaccinated, thousands of shipments take place and many other transactions are executed. All these activities need to be immediately processed by the system and specifically by all involved enterprises. A transportation company that distributes vaccines across the world might maintain its data in multiple data shards.

Existing blockchain solutions, however, are not able to meet all requirements of multi-enterprise collaboration workflows. While Caper [5] enables enterprises to keep their local data confidential, it does not address any of the **R1** to **R4** requirements. Fabric [12] addresses confidential collaboration using private data collection (although with a high overhead) and scalability using channels. However, Fabric does not support requirements **R2** and **R3**. Several variants of Fabric, such as Fast Fabric [45], Fabric++ [78], FabricSharp [75], and XOX Fabric [44], try to address the performance shortcomings of Fabric, especially when dealing with contentious workloads. However, these permissioned blockchain systems, similar to Fabric, suffer from the overhead of confidential collaboration and also do not support requirements **R2** and **R3**.

3 QANAAT MODEL

Qanaat is a permissioned blockchain system designed to support multi-enterprise applications. This section presents the Qanaat model and demonstrates how it supports different requirements of multi-enterprise applications.

3.1 Model Assumptions

Qanaat consists of a set of collaborative *enterprises*. Each enterprise owns a set of *nodes* (i.e., servers) that are grouped into different *clusters*. Each enterprise further partitions its data into multiple *data shards*. Each cluster of nodes maintains a *shard* of the enterprise data and processes transactions on that data shard.

Qanaat assumes the partially synchronous communication model. In the partially synchrony model, there exists an unknown global stabilization time (GST), after which all messages between correct nodes are received within some unknown bound Δ. Qanaat inherits the standard assumptions of previous permissioned blockchain systems, including a strong computationally-bounded adversary that can coordinate malicious nodes and delay communication to compromise service, an unreliable network that connects nodes and might drop, corrupt, or delay messages, and the existence of pairwise authenticated communication channels, standard digital signatures and public-key infrastructure (PKI). A message m signed by node i is denoted as $\langle m \rangle_{\sigma_i}$. Qanaat further uses threshold signatures where each node i holds a distinct private key that is used to create a signature share $\sigma\langle m \rangle_i$ for message m. A node can generate a valid threshold signature $\sigma\langle m \rangle$ for m given $n - f$ signature shares from distinct nodes. A collision-resistant hash function $D(.)$ is also used to map a message m to a constant-sized digest $D(m)$.

We next present the data model (Section 3.2) and the blockchain ledger (Section 3.3) of Qanaat, which together address collaboration confidentiality (**R1**) and data consistency (**R2**) requirements and then demonstrate the Qanaat infrastructure (Section 3.4) that addresses the confidential data leakage prevention requirement (**R3**). For simplicity of presentation, we first consider single-shard enterprises and then show how the model and the ledger are extended to support multi-shard enterprises (Section 3.5), which addresses the scalability requirement (**R4**).

3.2 Data Model

Qanaat constructs a hierarchical data model consisting of a set of *data collections* for each collaboration workflow. Each data collection is a separate datastore where the execution of transactions updates its data. Each data collection further has its own (business) logic to execute transactions. *Public* transactions of a collaboration workflow, e.g., `place order` in the vaccine supply chain, are executed on the *root* data collection. All enterprises maintain the root data collection. This is needed because enterprises use the data maintained in the root data collection in other transactions. For example, the `order` data stored in the root data collection is used by the supplier in its private transactions to provide raw materials.

Internal private transactions of each enterprise, on the other hand, are executed on its *local* data collection. Internal transactions are performed within an enterprise following the logic of the enterprise, e.g., `develop formulation`, and `package vaccines` take place within the pharmaceutical manufacturer.

Any subset of enterprises might also execute private transactions among themselves and confidentially from other enterprises. Such transactions are executed on a data collection shared among and maintained by *only* the involved enterprises. For example, the supplier executes private transactions with the pharmaceutical company on the material demand on a data collection that is maintained only by the supplier and pharmaceutical company.

Figure 2(a) presents a data model for a collaboration workflow with enterprises A, B, C, and D. At the top level, a public data collection d_{ABCD} is stored on all four enterprises. At the bottom level, there are four private data collections d_A, d_B, d_C, and d_D stored on the corresponding enterprises. The figure also includes all possible private data collections with different subsets of enterprises, e.g., d_{AB} or d_{ACD}. When a subset of enterprises, e.g., A and B, creates a data collection, e.g., d_{AB}, to execute private transactions, the shared data collection maintains the execution results of transactions that are executed on the data collection. Such data records are different

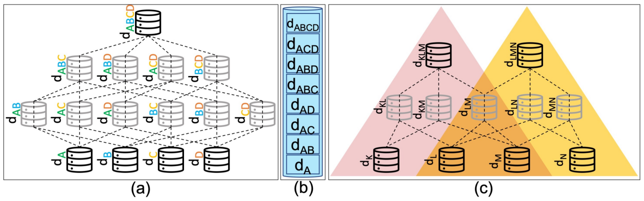

Figure 2: (a) A data model for a 4-enterprise collaboration workflow. Intermediate data collections are optional and needed only if there is a collaboration among a specific subset of enterprises, (b) data maintained by enterprise A, (c) data models for 3-enterprise workflows.

from the records maintained in the data collection of each individual enterprise, e.g., d_A and d_B. In contrast to the root and the local data collections that are needed in every collaboration workflow, intermediate data collections are *optional* and needed only if there is a collaboration among a specific subset of enterprises.

Qanaat defines operational primitives necessarily to capture data consistency and data dependency across data collections.

Write. The execution results of transactions executed on a data collection are written on the records of the same data collection. This is needed to guarantee collaboration confidentiality as (intermediate) data collections are created for private collaboration, e.g., private lobbying, among a subset of enterprises while preventing other enterprises from accessing the data. In particular, if enterprises A and B make a confidential collaboration, the resulting data should not be written on a data collection that is either shared with some other enterprise C or not shared with one of the involved enterprises, e.g., d_A. Note that d_{AB} is maintained by both enterprises A and B and they have access to its record. This captures the confidential collaborations in real-world cross-enterprise applications.

Read. When a transaction is being executed on a data collection d_X, it might read records of a data collection d_Y if enterprises sharing d_X are a subset of the enterprises sharing d_Y. For example, the internal transactions of the supplier can read the records of all data collections that the supplier is involved in with other enterprises. This is needed because those records might affect the internal transactions of the supplier, e.g., the number of vaccines that the supplier supplies depends on the orders it receives from other enterprises.

More formally, given a data collection d_X where X is the set of enterprises sharing d_X. For each data collection d_Y where $X \subseteq Y$, we define d_X as *order-dependent* of d_Y. We say transactions executed on d_X can read the records of d_Y if data collection d_X is order-dependent on d_Y. The dashed lines between data collections in Figure 2(a) present order-dependency among data collections; the transactions executed on the lower-level data collections can read the records of the higher-level data collections, e.g., d_{AB} can read d_{ABC}, d_{ABD} and d_{ABCD}.

In some applications, transactions that are executed on data collection d_X might also need to *verify* the records of another data collection d_Y in a *privacy-preserving* manner (i.e., without reading the exact records) if enterprises sharing d_X are a *superset* of the enterprises sharing d_Y, i.e., $Y \subset X$.

In particular, for intangible assets, e.g., cryptocurrencies, if enterprise A initiates a transaction in data collection d_{AB} that consumes some coins, enterprise B needs to verify the existence of the coins in data collection d_A. In supply chain workflows, however, since transactions are usually placed as a result of physical actions, verifying the records of order-dependent data collections is unnecessary. Qanaat can be extended to support privacy-preserving verifiability using advanced cryptographic primitives like secure multiparty computation [14, 40] and zero-knowledge proofs [42, 57, 61].

Every enterprise maintains all data collections that the enterprise is involved in, i.e., the root, a local, and perhaps several intermediate data collections. Qanaat replicates shared data collections on all involved enterprises to facilitate the use of shared data by the transactions on order-dependent data collections. Note that such shared data collections can be maintained by a third party, e.g., a cloud provider, trusted by all enterprises. However, this is a centralized solution that contradicts the decentralized nature of blockchains. Moreover, the data maintained in a shared data collection might be used (i.e., read) in transactions on all order-dependent data collections. Hence, enterprises need to query the cloud for every simple read operation to access the data.

As shown in Figure 2(b), enterprise A maintains local data collection d_A, root data collection d_{ABCD}, and all intermediate data collections (if exist), e.g., d_{AB} and d_{ACD}.

A data model represents data processed by each collaboration workflow among a set of enterprises. However, an enterprise (or a group of enterprises) might be involved in multiple collaboration workflows (instances of Qanaat) with different sets of enterprises. The transactions of the same enterprise in different collaboration workflows might be data-dependent. Creating an independent data collection for each enterprise in each collaboration workflow might result in *data consistency* issues. For example, a supplier that provides raw materials for both Pfizer and Moderna vaccines in two different supply chain collaboration workflows needs to know the total number of the requested materials in order to provision for them correctly. As a result, Qanaat creates a single data collection for each enterprise and if an enterprise is involved in multiple possibly data-dependent collaboration workflows, the transactions of the enterprise in different collaboration workflows are executed on the same data collection. An enterprise might also collaborate with enterprises outside Qanaat and require to access its data. Such data operations can be performed on the enterprise's local data collection without violating data consistency enabling the enterprise to use such data in its collaboration workflows within Qanaat.

Figure 2(c) presents two data models for two collaboration workflows where enterprises K, L, and M are involved in the first and enterprises L, M, and N are involved in the second collaboration workflow. Since L and M are involved in both workflows, the local data collections d_L and d_M and the intermediate data collection d_{LM} are shared in both collaboration workflows.

3.3 Blockchain Ledger

The blockchain ledger is an append-only data structure to maintain transaction records. When several enterprises execute transactions across different data collections, maintaining consistency and preserving confidentiality in a scalable manner is challenging. Before describing the Qanaat ledger, we discuss three possible solutions for ordering transactions in cross-enterprise collaborations.

1. A single, global ledger. One possibility is to construct a linear blockchain ledger for each collaboration workflow where all transactions on all data collections are totally ordered and appended to a single global ledger. To preserve the confidentiality of private transactions, the cryptographic hash of such transactions (instead of the actual transaction) can be maintained in the blockchain ledger. This technique, which has been used in Hyperledger Fabric [12], suffers from two main shortcomings. First, forcing *all* transactions into a single, sequential order unnecessarily degrades performance, especially in the large-scale collaborations targeted by Qanaat. To see this, note that while total ordering is needed for data-dependent transactions, e.g., transactions executed on the same data collection, to preserve data consistency, total ordering among transactions of independent data collections, e.g., d_A and d_B, is clearly not needed. Second, since this solution requires a single, global blockchain ledger to be maintained by every enterprise, each enterprise needs to maintain (the hash of) transactions that the enterprise was not involved in, e.g., internal transactions of other enterprises, which increases bandwidth and storage costs.

2. One ledger for each data collection. A second possibility is to maintain a separate transaction ordering (i.e., linear blockchain ledger) for each data collection. The blockchain ledger of each enterprise then consists of multiple parallel linear ledgers (one per each data collection that the enterprise is involved in). This solution, however, does not consider dependencies across order-dependent data collections where the execution of a transaction might require reading records from other data collections. Such dependencies need to be captured during the *ordering phase* to ensure that all replicas read the same state in the execution phase. Note that the read-set and write-set of transactions might not be known before execution. As a result, writing the read values into the block is not possible in the ordering phase.

For example, consider an internal transaction of the supplier that reads the order record of the root data collection and based on that, computes and stores the results. In the meantime (during the period from the initiation to the execution of the transaction), the value of the order record might change. As a result, if the state of the root data collection has not been captured, different nodes (i.e., replicas) of the supplier, might read different values (old or new) of the order in the execution phase resulting in inconsistency.

3. One ledger for each enterprise. A third solution is to maintain a total order among all transactions in which an enterprise is involved. This solution, in contrast to the second solution, guarantees data consistency. However, similar to the first solution, this solution prevents order-independent transactions from being appended to the ledger in parallel. For example, transactions on data collections d_{AB} have no data dependency with transactions on d_{AC} and can append to the ledger of enterprise A in parallel.

Qanaat addresses the shortcomings of the existing solutions in guaranteeing data consistency and preserving collaboration confidentiality while creating a ledger in an efficient and scalable manner. In Qanaat, the blockchain ledger guarantees two main properties:

- **Local consistency.** A total order on the transactions of each data collection is enforced due to data dependency among transactions executed on a data collection, and
- **Global consistency.** The order of the transactions of data collection d_X with respect to the transactions of all data collections that d_X is order-dependent on, is determined by capturing the state of such data collections. Moreover, transactions are ordered across the enterprises consistently.

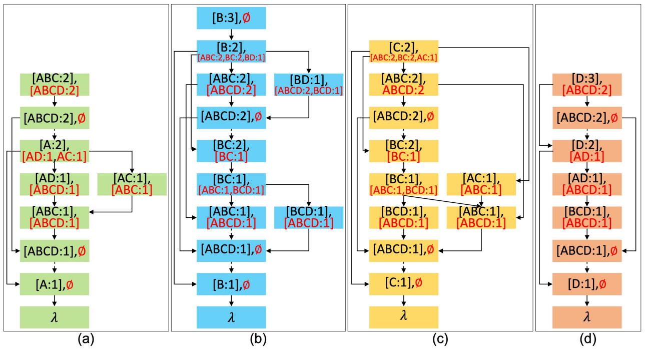

Figure 3: (a)-(d) The blockchain ledger for enterprises A, B, C, and D in a collaboration workflow

Qanaat constructs a single DAG-structured ledger for each enterprise consisting of transaction records of the data collections that are maintained by the enterprise. The first rule guarantees data consistency within each single data collection and the second rule ensures global data consistency across all order-dependent data collections maintained by an enterprise. Given the above rules, each transaction t initiated on data collection d_X has an identifier, ID = $\langle \alpha, \gamma \rangle$, composed of a *local* part α and possibly a *global* part γ where the ID is assigned during the ordering phase. The local part α is $[X{:}n]$ where X is a *label* representing the involved enterprises in d_X and n is a *sequence number* representing the order of transaction t on d_X. In the global part γ, for *every* data collection d_Y where d_X is order-dependent on d_Y, $Y{:}m$ is added to the γ of t to capture the state of data collection d_Y. Here, $Y{:}m$ is the local part of the ID of the last transaction that is committed on data collection d_Y. When t is executed, it might read the state of a subset of data collections that are captured in the global part of its ID.

Figure 3(a)-(d) shows the blockchain ledger of four enterprises A, B, C, and D created in the Qanaat model (following Figure 2(a)) for a collaboration workflow. In this figure, $\langle [A{:}1], \varnothing \rangle$ is an internal transaction on data collections d_A (with $\gamma = \varnothing$ because no transactions has been processed yet). $\langle [ABCD{:}1], \varnothing \rangle$ is a public transaction on d_{ABCD} (with $\gamma = \varnothing$ because the root data collection is not order-dependent on any data collections). As shown, two transactions $\langle [ABC{:}1], [ABCD{:}1] \rangle$ (on d_{ABC}) and $\langle [BCD{:}1], [ABCD{:}1] \rangle$ (on d_{BCD}) are appended to the ledger of enterprise B (as well as C) in parallel because d_{ABC} and d_{BCD} are not order-dependent. Both transactions include global part $\gamma = [ABCD{:}1]$ because both d_{ABC} and d_{BCD} are order-dependent on d_{ABCD}.

3.4 Qanaat Infrastructure

In Qanaat, nodes follow either the crash failure model or the Byzantine failure model. To guarantee *fault tolerance*, clusters with crash-only nodes include $2f{+}1$ nodes. In the presence of Byzantine nodes, $3f{+}1$ nodes are needed to provide fault tolerance [22]. The malicious nodes, however, can violate data confidentiality by leaking requests, replies, and data stored and processed at the nodes. To prevent confidential data leakage (known as *intrusion tolerance*

[54, 85]), Qanaat can use either of the following two techniques: (1) restricting the data that nodes can access [19, 20, 62, 68, 85] using secret sharing schemes, or (2) adding a privacy firewall between the ordering nodes and the execution nodes [39, 86].

In the secret sharing scheme, clients encode data using an $(f + 1, n)$-threshold secret sharing scheme, where $f + 1$ shares out of n total shares are needed to reconstruct the confidential data [54]. Secret sharing schemes only perform basic store and retrieve operations (Belisarius [68] also supports addition) and do not support general transactions that require nodes to manipulate the contents of stored data. As a result, this technique is not suitable for blockchain systems that are supposed to support complex transactions.

In the privacy firewall mechanism [86], the infrastructure consists of $3f + 1$ ordering nodes (where f is the maximum number of malicious ordering nodes) that run a BFT protocol to order client requests, $2g + 1$ execution nodes (where g is the maximum number of malicious execution nodes) that maintain data and deterministically execute arbitrary transactions following the order assigned by ordering nodes, and a privacy firewall consisting of a set of $h + 1$ rows of $h + 1$ filters (where h is the maximum number of malicious filter nodes) between the ordering nodes and execution nodes. The privacy firewall architecture assumes a network configuration that physically restricts communication paths between ordering nodes, filters, and execution nodes, i.e., clients can only communicate with ordering (and not execution) nodes, and each filter has a physical network connection only to all (filter) nodes in the rows above and below. As a result, a malicious node can *either* access confidential data (an execution node or a filter) *or* communicate freely with clients (an ordering node) *but not both*. The $h+1$ rows of $h+1$ filters guarantee that first, there is at least one path between execution nodes and ordering nodes which only consists of non-faulty filters (liveness) and second, since there are $h + 1$ rows and maximum h malicious filters, there exist a row consisting of only non-faulty filters that filters any malicious message (possibly including confidential data). As a result, the rows below have no access to any confidential data leaked by malicious execution nodes. Note that if $h \leq 3f$, ordering nodes can be merged with the bottom row of filters by placing a filter on each ordering node. In most applications, request and reply bodies must also be encrypted, thus, ordering nodes cannot read them (while clients and execution nodes can).

By separating ordering nodes from execution nodes, a simple majority of non-faulty nodes is sufficient to mask Byzantine failure among execution nodes, i.e., $2g + 1$ execution nodes can tolerate g Byzantine faults. This is important because, compared to ordering, executing transactions, and maintaining the application logic (e.g., smart contracts) and data require more powerful computation, storage, and I/O resources. The overhead of the privacy firewall mechanism can be reduced by adding h filters to each row while providing a higher degree of confidentiality [39].

Separating ordering nodes from execution nodes and using a privacy firewall comes with extra resource costs because ordering nodes and execution nodes need to be physically separated. However, if a cluster is deployed in a cloud platform, ordering nodes and execution nodes can match the control layer and computing nodes, respectively. Similarly, tools such as internal authorization services, node auditors, and load balancers deployed in existing cloud platforms can be used as privacy firewalls [39].

Qanaat prevents confidential data leakage despite Byzantine failure using the privacy firewall mechanism presented in [86].

3.5 Multi-Shard Enterprises

We now show how the data model and the blockchain ledger can be extended to support multi-shard enterprises. With single-shard enterprises, every execution node of each enterprise maintains all data collections that the enterprise is involved in. Enterprises, however, partition their data into different shards. Each data shard is then assigned to a cluster of nodes where ordering nodes of the cluster order the transactions and execution nodes maintain the data shard and execute transactions on the data shard.

We assume that enterprises use the same sharding schema for each shared data collection to facilitate transaction processing across different enterprises. The schema is agreed upon by all involved enterprises when a data collection is created, i.e., the sharding schema is part of the configuration metadata. Using the same sharding schema leads to a more efficient ordering phase for cross-enterprise transactions because there is no need for all involved enterprises to run a consensus protocol to agree on the order of every transaction. For each cross-enterprise transaction, one enterprise orders the transaction and other involved enterprises only validate the order (details in Sections 4.3 and 4.4). Furthermore, different data shards of an enterprise are processed by different clusters. Using the same sharding schema, enterprises can easily communicate with the right cluster that processes the data records of transactions. If enterprises use different sharding schemas for a shared data collection, Qanaat could still process cross-enterprise transactions but with the overhead required to execute a consensus protocol with every transaction on each cluster.

The blockchain ledger of a single-shard enterprise maintains all transactions that are executed on the enterprise data. In a multi-shard enterprise, the enterprise data is partitioned into different shards where each shard is replicated on a cluster of execution nodes. Since each cluster maintains a separate data shard, it executes a different set of transactions. As a result, each cluster in a multi-shard enterprise needs to maintain a different ledger. The ledger of each cluster of a multi-shard enterprise, however, is constructed in the same way as a single-shard enterprise. Moreover, the notion of global consistency is extended to guarantee that each cross-shard transaction is ordered across participating shards consistently.

4 TRANSACTION PROCESSING IN QANAAT

Processing transactions requires establishing consensus on a unique order of requests. Fault-tolerant protocols use the State Machine Replication (SMR) technique [59, 77] to assign each client request an order in the global service history. In an SMR fault-tolerant protocol, all non-faulty nodes execute the same requests in the same order (*safety*) and all correct client requests are eventually executed (*liveness*). Qanaat guarantees safety in an asynchronous network, however, it considers a synchrony assumption to ensure liveness (due to FLP results [41]).

A transaction might be executed on a single shard or on multiple shards of a data collection where the data collection belongs to a single enterprise (i.e., a local data collection) or is shared among multiple enterprises (i.e., a root or an intermediate data collection).

Table 1: Processing transactions in Qanaat

Transaction Type		Example	Consensus Protocol	
Shard	Enterprise	Clusters- Shards	Coordinator-based	Flattened
intra	intra	A_1 - d^1_A	Pluggable	Pluggable
intra	cross	C_3, D_3 - d^3_{CD}	Fig. 4(a)	Fig. 5(a)
cross	intra	A_2, A_3 - d^2_A, d^3_A	Fig. 4(b)	Fig. 5(b)
cross	cross	B_1,C_1,B_2,C_2- d^1_{BC},d^2_{BC}	Fig. 4(c)	Fig. 5(c)

As a result, as shown in Table 1, Qanaat needs to support four different types of transactions. In Table 1, the network consists of four enterprises where each enterprise $\psi \in \{A, B, C, D\}$ owns three clusters of nodes, e.g., ψ_1, ψ_2 and ψ_3 and processes shard d^i_ψ by nodes of cluster ψ_i.

An *intra-shard* transaction is executed on the same shard of a data collection either *intra-enterprise*, e.g., on shard d^1_A of a local data collection d_A by cluster A_1, or *cross-enterprise*, e.g., on shard d^3_{CD} of a shared data collection d_{CD} by clusters C_3 and D_3. A *cross-shard transaction* is executed on multiple shards of a data collection either *intra-enterprise*, e.g., on shards d^2_A and d^3_A of a data collection d_A by clusters A_2, A_3, or *cross-enterprise*, e.g., on shards d^1_{BC} and d^2_{BC} of a shared data collection d_{BC} by clusters B_1, C_1, B_2, and C_2.

In Qanaat, cross-shard intra-enterprise transactions and intra-shard cross-enterprise transactions are handled differently. In a cross-shard intra-enterprise transaction, different clusters maintain separate data shards. As a result, each cluster needs to separately reach agreement on the transaction order among transactions of its shard. However, in an intra-shard cross-enterprise transaction, since the involved enterprises use the same sharding schema for each shared data collection, the cluster that initiates the transaction and all the involved clusters process the transaction on the same shard of data. Hence, the order of transactions on different clusters is the same. As a result, it is sufficient that one (initiator) cluster determines the order and other clusters only validate the suggested order. Note that while the involved clusters belong to different distrustful enterprises, all clusters can detect any malicious behavior of the initiator cluster. We will discuss how Qanaat addresses all possible malicious behaviors, e.g., assigning invalid ID (order) or even not sending the transaction to other clusters. In cross-enterprise collaboration, enterprises cannot trust that the other enterprise nodes might not be compromised. As a result, independent of the declared failure model of nodes, Qanaat uses a BFT protocol to order cross-enterprise transactions. For cross-shard intra-enterprise transactions, on the other hand, the involved clusters belong to the same enterprise and trust each other. As a result, if all nodes are crash-only, cross-shard consensus can be achieved using a crash fault-tolerant protocol.

To process transactions across clusters *coordinator-based* and *flattened* approaches are used. In the coordinator-based approach, inspired by existing permissioned blockchains such as AHL [35], Saguaro [10], and Blockplane [65], a cluster coordinates transaction ordering, whereas, in the flattened approach, transactions are ordered across the clusters without requiring a coordinator.

Nodes in Qanaat follow different failure models, i.e., crash or Byzantine. We mainly focus on two common scenarios. First, when nodes follow the crash failure, a cluster contains $2f + 1$ nodes that perform both ordering and execution. Second, when nodes follow the Byzantine failure and each cluster includes $3f + 1$ ordering nodes, $2g + 1$ execution nodes, and a privacy firewall consisting of $h + 1$ rows of $h + 1$ filters. The number of required matching votes to ensure that a quorum of ordering nodes within a cluster agrees with the order of a transaction is different in different settings. We define *local-majority* as the number of required matching votes from a cluster. For crash-only clusters, local-majority is $f + 1$ (from $2f + 1$ nodes), and for clusters with Byzantine ordering nodes, local-majority is $2f + 1$ (from $3f + 1$ ordering nodes).

4.1 Intra-Cluster Consensus

In the internal (intra-cluster) consensus protocol, ordering nodes of a cluster, *independently* of other clusters, agree on the order of a transaction. The internal consensus protocol is pluggable and depending on the failure model of nodes of the cluster a crash fault-tolerant (CFT) protocol, e.g., (Multi-)Paxos [60] or a Byzantine fault-tolerant (BFT) protocol, e.g., PBFT [27], can be used. If nodes follow the Byzantine failure model, as discussed before, Qanaat separates ordering nodes from execution nodes and uses a privacy firewall to prevent confidential data leakage.

The protocol is initiated by a pre-elected ordering node of the cluster, called *the primary*. When the primary node $\pi(P_i)$ of cluster P_i receives a valid signed request message $\langle \text{REQUEST}, op, t_c, c \rangle_{\sigma_c}$ from an authorized client c (with timestamp t_c) to execute (encrypted) operation op on local data collection d_{P_i}, it initiates the protocol by multicasting a message, e.g., accept message in Multi-Paxos (assuming the primary is elected) or pre-prepare message in PBFT, including the request and its digest to other ordering nodes of the cluster.

To provide a total order among transaction blocks and preserve data consistency, the primary also assigns an ID, as discussed in Section 3.3, to the request. The ordering nodes of the cluster then agree on the order of the transaction using the utilized protocol.

4.2 Transaction Execution Routine

The next step after ordering is to execute a transaction and inform the client. If nodes follow the Byzantine failure model, ordering and execution are performed by distinct sets of nodes that are separated by a privacy firewall. Once a transaction is ordered, the ordering nodes generate a commit certificate consisting of signatures from $2f + 1$ different nodes and multicast both the request and the commit certificate to the bottom row of filters. The filters check the request and the commit certificate to be valid and multicast them to the next row above. Filters also track the requests that they have seen in a commit or a reply certificate and their IDs.

Upon receiving both a valid request and a valid commit certificate, execution nodes append the transaction and the corresponding commit certificate to the ledger. Note that the execution nodes store the last reply certificate sent to client c to check if the current request is new (i.e., has a larger timestamp) that needs to be executed and committed to the ledger or an old request (i.e., has the same or smaller timestamp) where nodes re-send the reply messages. If all transactions accessing on d_{P_i} with lower local sequence numbers ($< n$) have been executed, nodes execute the transaction on data collection d_{P_i} following its application logic. If, during the execution, the execution nodes need to read values from some other data collection d_X where d_{P_i} is order-dependent on d_X (i.e., $P_i \subseteq X$),

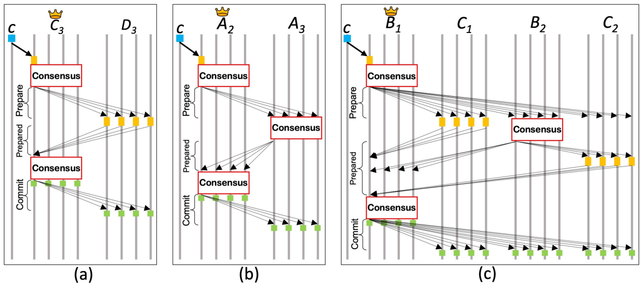

Figure 4: (a) intra-shard cross-enterprise, (b) cross-shard intra-enterprise, and (c) cross-shard cross-enterprise coordinator-based consensus protocols

the nodes check the public part of the transaction ID to ensure that they read the correct state of d_X. Data collections store data in multi-versioned datastores to enable nodes to read the version they need to. This ensures that all execution nodes execute requests in the same order and on the same state. Execution nodes then multicast a signed reply message including the (encrypted) results to the top row of filters. When a filter node in the top row receives $g + 1$ valid matching reply messages, it generates a reply certificate authenticated by $g + 1$ signature shares from distinct execution nodes and multicasts it to the row below. Each filter then multicasts the reply certificate to the row of filter nodes or ordering nodes below. Finally, client c accepts the result once it receives a valid reply certificate from ordering nodes.

If nodes follow the crash failure model, ordering and execution are performed by the same set of nodes. In this case, nodes append the transaction to the ledger and execute it as explained before and then the primary node sends a reply message to client c.

4.3 Coordinator-based Consensus

The coordinator-based approach has been used in distributed databases to process cross-shard transactions, e.g., two-phase commit. Coordinator-based protocols have also been used in several permissioned blockchain systems, e.g., AHL [35], Blockplane [65], and Saguaro [10]. This section briefly demonstrates how Qanaat can leverage the coordinator-based approach to process cross-cluster transactions. In contrast to the existing coordinator-based databases and blockchains that address multi-shard single-enterprise contexts, Qanaat deals with multi-shard multi-enterprise environments. Moreover, since each transaction might read data from multiple data collections, ordering transactions is Qanaat is more complex. Figure 4 presents the normal case operation of (a) intra-shard cross-enterprise, (b) cross-shard intra-enterprise, and (c) cross-shard cross-enterprise coordinator-based protocols. All three coordinator-based protocols consist of prepare, prepared, and commit phases.

In the prepare phase and upon receiving a transaction, ordering nodes of the *coordinator* cluster establish (intra-cluster) consensus on the order of the transaction among the transactions of their shard. The primary then sends a signed prepare message to the ordering nodes of all involved clusters.

The prepared phase is handled differently in different protocols. For intra-shard cross-enterprise transactions, all involved clusters, e.g., C_3 and D_3 in Figure 4(a), maintain the same data shard. As a result, there is no need to run consensus in non-coordinator clusters, e.g., D_3. Upon receiving a prepare message from the primary of the coordinator cluster, e.g., C_3, each ordering node of a non-coordinator cluster, e.g., D_3, validates the order and sends a prepared message to the primary of the coordinator cluster.

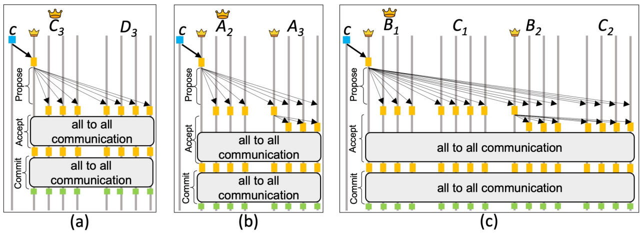

Figure 5: (a) intra-shard cross-enterprise, (b) cross-shard intra-enterprise, and (c) cross-shard cross-enterprise flattened consensus protocols

For cross-shard intra-enterprise transactions, involved clusters, e.g., A_2 and A_3, maintain different shards of a local data collection, e.g., d_A. Upon receiving a prepare message from the coordinator cluster, e.g., A_2, each non-coordinator cluster separately establishes consensus on the order of the transaction in its data shard. The primary of each non-coordinator cluster, e.g., A_3, then sends a prepared message to the nodes of the coordinator cluster, e.g., A_2.

Finally, for cross-shard cross-enterprise transactions, involved clusters of the initiator enterprise, e.g., B_1 and B_2 of enterprise B, maintain different shards of a shared data collection, e.g., d_{BC}, while each shard is replicated on clusters of different enterprises, e.g., d^1_{BC} is maintained by B_1 and C_1. Upon receiving a prepare message from the coordinator cluster, e.g., B_1, if the cluster belongs to the initiator enterprise, e.g., B_2, it establishes (internal) consensus on the transaction order. Otherwise, the cluster waits for a message from the cluster of the initiator enterprise that maintains the same data shard as they do, e.g., C_1 waits for B_1 and C_2 waits for B_2. Each node then validates the received message and sends a prepared message to the primary of the coordinator cluster.

In the commit phase, upon receiving valid prepared messages from every involved cluster, the primary of the coordinator cluster establishes internal consensus within its cluster and multicasts a signed commit message to ordering nodes of all involved clusters.

Each cluster then uses the transaction execution routine (Section 4.2) to execute the transaction and send reply back to the client.

4.4 Flattened Consensus

The flattened consensus protocols of Qanaat, in contrast to the flattened cross-shard consensus protocols, e.g., SharPer [8], which are designed for single-enterprise environments, support multiple multi-shard collaborative enterprises.

The flattened protocols of Qanaat enable clusters to commit a transaction in a smaller number of communication phases. For instance, a cross-shard intra-enterprise transaction can be committed using the flattened protocol in 3 cross-cluster communication steps. In contrast, using the coordinator-based protocol, the same transaction requires 3 cross-cluster and 9 intra-cluster communication steps (3 runs of PBFT). It also reduces the load on the coordinator cluster. We now discuss flattened consensus protocols in detail.

4.4.1 Intra-Shard Cross-Enterprise Consensus. In the intra-shard cross-enterprise protocol, multiple clusters from different enterprises process a transaction on the same data shard of a shared data collection, e.g., clusters C_3 and D_3 on shard d^3_{CD} of shared data collection d_{CD} (Figure 5(a)). Upon receiving a valid request message m, the primary ordering node $\pi(P_i)$ of the initiator cluster P_i assigns transaction ID (as discussed in Section 3.3) and multicasts a signed propose message $\langle \text{PROPOSE}, \text{ID}, d, m\rangle_{\sigma_{\pi(P_i)}}$ to the ordering nodes of all involved clusters where $d = D(m)$ is the digest of request m. Upon receiving a propose message, each node r validates the digest, signature and ID, and multicasts a signed accept message $\langle \text{ACCEPT}, \text{ID}, d\rangle_{\sigma_r}$ to all ordering nodes of every involved cluster. Each node r waits for valid matching accept messages from a local-majority of every involved cluster and then multicasts a commit message including the transaction ID and its digest to every ordering node of all involved clusters. The propose and accept phases of the protocol, similar to pre-prepare and prepare phases of PBFT, guarantee that non-faulty nodes agree on the transaction order. Finally, once an ordering node receives valid commit messages from the local-majority of all involved clusters that match its commit message, it generates a commit certificate by merging messages from all involved clusters. Each cluster uses the transaction execution routine to execute the transaction and send reply back to the client.

4.4.2 Cross-Shard Intra-Enterprise Consensus. In the cross-shard intra-enterprise consensus protocol, as shown in Figure 5(b), the involved clusters, e.g., A_2 and A_3, maintain different shards of a local data collection, e.g., d_A. Upon receiving a valid request message m, the primary $\pi(P_i)$ of the initiator cluster P_i assigns ID_i to the transaction and multicasts a propose message $\mu = \langle \text{PROPOSE}, \text{ID}_i, d, m\rangle_{\sigma_{\pi(P_i)}}$ to all nodes of every involved cluster. Upon receiving a valid propose message, each primary node $\pi(P_j)$ of an involved cluster P_j multicasts $\langle \text{ACCEPT}, \text{ID}_i, \text{ID}_j, d, d_\mu\rangle_{\sigma_{\pi(P_j)}}$ to the nodes of its cluster where ID_j represents the order of m on data shard of cluster P_j. The digest of the propose message, $d_\mu = D(\mu)$, is added to link the accept message with the corresponding propose message enabling nodes to detect changes and alterations to any part of the message. The primary of the initiator and all involved clusters process the request only if there is no (concurrent) uncommitted request m' where m and m' intersect in at least 2 shards. This is needed to ensure that requests are committed in the same order in different shards (global consistency) [8].

Each node r (including the primary) of an involved cluster P_j then multicasts a signed accept message $\langle \text{ACCEPT}, \text{ID}_i, \text{ID}_j, d, r\rangle_{\sigma_r}$ to all ordering nodes of every involved cluster. Upon receiving valid matching accept messages from the local-majority of all involved clusters $P_i, P_j, ..., P_k$, node r multicasts $\langle \text{COMMIT}, \text{ID}_i, \text{ID}_j, ..., \text{ID}_k, d, r\rangle_{\sigma_r}$ message to every ordering node of all involved clusters.

Upon receiving valid matching commit messages from the local-majority of every involved cluster, the node generates a commit certificate. The transaction execution routine then executes the transaction and appends the transaction to the ledger.

In this protocol, since all clusters belong to the same enterprise, if nodes of all involved clusters are crash-only, Qanaat processes transactions more efficiently. In that case, nodes of each involved cluster send the accept message to *only* the initiator primary and the initiator primary multicasts a commit message to all nodes (instead of the two all to all communication phases).

4.4.3 Cross-Shard Cross-Enterprise Consensus. The cross-shard cross-enterprise consensus protocol is needed when a transaction is executed on different shards of a non-local data collection shared between different enterprises, e.g., a transaction that is executed on data shards d^1_{BC} and d^2_{BC} of shared data collection d_{BC} where d^1_{BC} is maintained by B_1 and C_1 and d^2_{BC} is maintained by B_2, and C_2 (Figure 5(c)).

In this case, different clusters of each enterprise maintain different data shards, e.g., B_1 and B_2 maintain d^1_{BC} and d^2_{BC}, however, each data shard is processed by only one cluster of each involved enterprise, e.g., d^1_{BC} is maintained by B_1 and C_1. In Qanaat, in order to improve performance, only the clusters of the initiator enterprise, e.g., B_1 and B_2, establish consensus on the transaction order and clusters of other enterprises, e.g., C_1 and C_2, validate the order.

The primary node $\pi(P_i)$ of the initiator cluster P_i multicasts a signed propose message $\mu = \langle \text{PROPOSE}, \text{ID}_i, d, m\rangle_{\sigma_{\pi(P_i)}}$ to the nodes of all involved clusters. Upon receiving a valid propose message, each primary node $\pi(P_j)$ of an involved cluster P_j belonging to the initiator enterprise assigns an ID_j to the transaction and multicasts a signed $\langle \text{ACCEPT}, \text{ID}_i, \text{ID}_j, d, d_\mu\rangle_{\sigma_{\pi(P_j)}}$ message to all the nodes of its cluster *and* all other clusters that maintain the same data shard. As before, primary nodes ensure consistency (i.e., no concurrent request with more than one shared shard) before processing a request. The accept and commit phases process the transaction in the same way as cross-shard intra-enterprise protocol.

4.4.4 Primary Failure Handling. We use the retransmission routine presented in [86] to handle unreliable communication between ordering nodes and execution nodes. In this section, we focus on the failure of the primary ordering node. If the timer of node r expires before it receives a commit certificate, it suspects that the primary might be faulty. When the primary node of a cluster fails, the primary failure handling routine of the internal consensus protocol, e.g., view-change in PBFT, is triggered by timeouts to elect a new primary.

The timeout mechanism prevents the protocol from blocking and waiting forever by ensuring that eventually (after GST), communication between non-faulty ordering nodes is timely and reliable. Ordering nodes use different timers for intra-cluster and cross-cluster transactions because processing transactions across clusters usually takes more time. The timeout value mainly depends on the network latency (within or across clusters). It must be long enough to allow nodes to communicate with each other and establish consensus, i.e., at least 3 times the WAN round-trip for cross-cluster transactions to allow a view-change routine to complete without replacing the primary node again. The timeouts are adjusted to ensure that this period increases exponentially until some transaction is committed, e.g., in case of consecutive primary failure, the timeout is doubled as in PBFT [27].

While in the intra-shard cross-enterprise consensus, only the failure of the initiator primary needs to be handled, in other protocols, the primary of any involved clusters that assign an ID, i.e., belongs to the initiator enterprise, might fail. In the intra-shard cross-enterprise consensus, if the timer of node r in the initiator cluster expires, it initiates the failure handling routine of the internal consensus protocol, e.g., view-change in PBFT. Otherwise, if the node is in some other involved cluster, it multicasts a commit-query message including the request ID and its digest to the nodes of the initiator cluster. If nodes receive commit-query from the local-majority of an involved cluster for a request, they suspect their primary is faulty. Nodes also log the query messages to detect denial-of-service attacks initiated by malicious nodes.

In the second and third protocols, each transaction needs to be ordered by the initiator primary as well as the primary nodes of all clusters belonging to the initiator enterprise (i.e., which maintain different data shards). Suppose a malicious primary node (from the initiator or an involved cluster) sends a request with inconsistent ID to different nodes. In that case, nodes detect the failure in all to all communication phases of the protocol triggering the primary failure handling routine within the faulty primary cluster. Note that since propose and accept messages need to be multicast to all nodes, and all nodes of all involved clusters multicast accept and commit messages to each other, even if a malicious primary decides not to multicast a request to a group of nodes, it will be easily detected.

Finally, if a client does not receive a reply soon enough, it multicasts its request message to all ordering nodes of the cluster that it has already sent its request. If an ordering node has both the commit certificate (i.e., the request has already been ordered) and the reply certificate (i.e., the request has already been executed), the node simply sends the reply certificate to the client. If the node has not received the reply certificate, it re-sends the commit certificate to the filter nodes. Otherwise, if the node is not the primary, it relays the request to the primary. If the nodes do not receive propose messages, the primary will be suspected to be faulty.

We establish the safety (agreement, validity, and consistency) and liveness properties of both the coordinator-based and the flattened protocols in the extended version of the paper [11].

5 EXPERIMENTAL EVALUATION

Our evaluation has three goals: (1) comparing the performance of the coordinator-based and flattened consensus protocols in different workloads with different types of transactions; (2) demonstrating the overhead of using the privacy firewall mechanism [86] to provide confidentiality despite Byzantine faults; (3) comparing the performance of Qanaat with Hyperledger Fabric [12] and two of its variants, FastFabric [46] and Fabric++ [78], to analyze the impact of sharding and the performance trade-off between a higher number of shards versus a higher percentage of cross-shard transactions.

We analyze the impact of (1) the failure model of nodes, (2) the percentage of cross-enterprise transactions, (3) the percentage of cross-shard transactions, (4) the geo-distribution of clusters, (5) the number of involved enterprises, (6) the failure of nodes, and (7) the degrees of contention on the performance of these protocols and systems. Due to space restrictions, the experiment results for (5), (6) and (7) are reported in the extended version of the paper [11].

We implemented a prototype of Qanaat and deployed a simple asset management collaboration workflow. We use (modified) SmallBank benchmarks with write-heavy workloads. The key selection distribution in each data collection is uniform with s-value = 0.

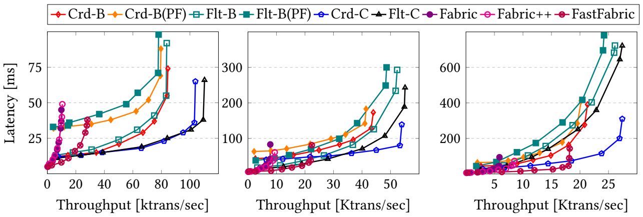

Figure 6: Workloads with (a) 10%**, (b)** 50%**, (c)** 90% **intra-shard cross-enterprise transactions**

We consider an infrastructure with 4 enterprises. Each enterprise partitions its data into 4 shards. Each crash-only clusters includes $2f + 1$ nodes that order and execute transactions while each Byzantine cluster includes $3f+1$ ordering nodes, $2g+1$ execution nodes and $h+1$ rows of $h+1$ filter nodes. To demonstrate the overhead of the privacy firewall mechanism, we also measure the performance of Byzantine clusters with only $3f+1$ nodes that order and execute transactions. In all experiments $f = g = h = 1$. We use Paxos and PBFT as the internal consensus protocols.

We consider a single-channel Fabric deployment ($v2.2$) where Raft [66] is its consensus protocol. We deploy 4 enterprises on the channel where enterprises can form public or private collaboration. Sharding was not possible in a single-channel Fabric deployment, however, we defined four endorsers to execute transactions of enterprises in parallel. We use a similar setting for FastFabric and Fabric++. FastFabric re-architects Fabric and provides efficient optimizations such as separating endorsers from storage nodes and sending transaction hashes to orderers. Fabric++, on the other hand, presents reordering and early abort mechanisms to improve the performance of Fabric, especially in contentious workloads.

The experiments were conducted on the Amazon EC2 platform. Other than the fourth set of experiments where clusters are distributed over 4 different AWS regions, clusters are placed in the same data center (California) with < 1 ms ping latency in the first three sets of experiments. Each VM is a c4.2xlarge instance with 8 vCPUs and 15GB RAM, Intel Xeon E5-2666 v3 processor clocked at 3.50 GHz. The results reflect end-to-end measurements from the clients. The reported results are the average of five runs.

5.1 Intra-Shard Cross-Enterprise Transactions

In the first set of experiments, we measure the performance of all protocols with different percentages, i.e., 10%, 50%, and 90%, of intra-shard cross-enterprise transactions. Each transaction is randomly initiated on a single data shard of a data collection shared among multiple enterprises. The number of involved enterprises depends on the data collection. Figure 6(a) demonstrate the results for the workload with 10% intra-shard cross-enterprise (and 90% internal) transactions. In this scenario and with crash-only nodes, Qanaat processes more than 110 ktps with 38 ms latency using the flattened protocol (Flt-C) and 103 ktps with 36 ms latency using the coordinator-based protocol (Crd-C).

Fabric processes only 9.7 ktps (8% of the throughput of Flt-C) with 37 ms latency. While different endorsers of different enterprises execute their transactions in parallel, ordering the transactions of all enterprises by a single set of orderers becomes a bottleneck. This

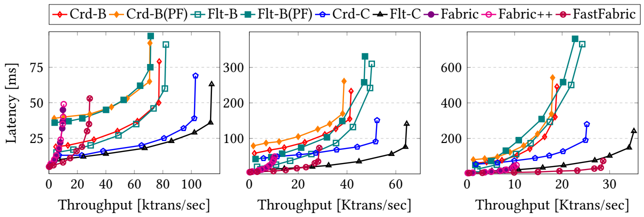

Figure 7: Workloads with (a) 10%**, (b)** 50%**, (c)** 90% **cross-shard intra-enterprise transactions**

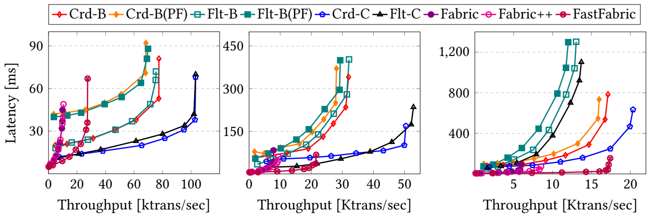

Figure 8: Workloads with (a) 10%**, (b)** 50%**, (c)** 90% **cross-shard cross-enterprise transactions**

clearly demonstrates the impact of parallel ordering (due to sharding) in Qanaat, where different clusters process their transactions independently. Fabric++ demonstrates only 3% higher throughput compared to Fabric with the same latency as it can reorder and early abort invalidated transactions. FastFabric, however, demonstrates 189% throughput improvement compared to Fabric due to its optimized architecture. However, its throughput is still 26% of the throughput of Flt-C with the same latency.

With Byzantine nodes, the performance of the flattened protocol (Flt-B) and the coordinator-based protocol (Crd-B) is reduced, which is expected due to the higher complexity of BFT protocols. Using the privacy firewall mechanism results in 8% and 6% lower throughput in the coordinator-based (Crd-B(PF)) and flattened (Flt-B(PF)) protocols respectively. This slight throughput reduction is the result of two infrastructural changes. On one hand, using the privacy firewall mechanism, request and reply messages go through filters resulting in lower performance. On the other hand, separating ordering from execution reduces the performance overhead by decreasing the load on ordering nodes. The privacy firewall mechanism also increases the latency of transaction processing in different workloads by a constant coefficient. This is expected because the increased latency comes from sending messages through the filters, which is separated from the consensus routine; the bottleneck in heavy workloads. While this latency is considerable in light workloads, it becomes much lower in heavy workloads, e.g., 166% vs. 25% higher latency in Crd-B(PF) protocol.

Increasing the percentage of cross-enterprise transactions to 50%, as shown in Figure 6(b), reduces the throughput of all protocols. This is expected because a higher percentage of transactions requires cross-enterprise consensus. In this experiment, Flt-B processes 52 ktps with 230 ms latency while Crd-B processes 43 ktps (18% lower) with 130 ms latency (44% lower). This shows a trade-off between the number of communication phases and the quorum size. While the coordinator-based approach requires more phases of message passing (resulting in lower throughput), the quorum size of the flattened approach is larger, i.e., all nodes communicate with each other (resulting in higher latency). Using the privacy firewall mechanism, the throughput of Crd-B(PF) and Flt-B(PF) is increased by 5% and 7% respectively and their latency is increased by only 9% and 6% (compared to Crd-B and Flt-B) before the end-to-end throughput is saturated. These results demonstrate that as the ordering routine becomes heavily loaded, the overhead of using the privacy firewall mechanism is alleviated.

The performance of Crd-C is significantly better than Crd-B (i.e., 23% higher throughput, 39% lower latency) in this scenario. This difference, however, is not remarkable in the flattened protocols. The reason is that in the coordinator-based protocol, consensus takes place within each cluster using the internal consensus protocols (Paxos in this case). However, in the flattened protocol, as shown in Figure 5(a), there is no internal consensus within clusters and a BFT protocol across enterprises establishes agreement.

The performance of Fabric is also affected by increasing the percentage of cross-enterprise transactions due to (1) the overhead of hashing techniques and (2) conflicting transactions [6, 44, 45, 78]. Interestingly, the throughput gap between Fabric and Fabric++ is increased to 18% (from 3%) in this scenario due to early abort and reordering mechanisms presented in Fabric++.

With 90% cross-enterprise transactions, as shown in Figure 6(c), the latency of Flt-B becomes very high (680 ms) due to its $O(n^2)$ message communication. The performance of Flt-C and Flt-B becomes close to each other since in both cases, a BFT protocol is used for cross-enterprise consensus. In this experiment, FastFabric demonstrates the lowest latency (35% lower than Crd-C with 18 ktps) as it does not need message communication across clusters.

5.2 Cross-Shard Intra-Enterprise Transactions

In the second set of experiments, we measure the performance of coordinator-based and flattened cross-shard intra-enterprise protocols. With Byzantine nodes and with 10% cross-shard transactions, as shown in Figure 7(a), the performance of Crd-B is still close to Flt-B. However, by increasing the percentage of cross-shard transactions to 50%, Flt-B shows 20% higher throughput. In this set of experiments, since all shards belong to a single enterprise, as explained in Section 4.4.2, Flt-C is implemented as a CFT protocol, and, as shown in Figure 7(a)-(c), has the best performance in all three workloads. Similar to the previous section, the overhead of using the privacy firewall mechanism is alleviated when the ordering nodes become highly loaded, e.g., the gap between the latency of Flt-B(PF) and Flt-B is reduced from 25% to 4% by increasing the percentage of cross-shard transactions from 10% to 90%. Since enterprises maintain their data on a single data shard, Fabric demonstrates the same performance in all three workloads, which is significantly worse than Qanaat. However, with 90% cross-shard transactions, similar to cross-enterprise transactions, FastFabric shows the lowest latency.

5.3 Cross-Shard Cross-Enterprise Transactions

In workloads consisting of cross-shard cross-enterprise transactions, as shown in Figure 8(a)-(c), the coordinator-based protocol shows better performance, especially with a higher percentage of cross-cluster transactions. In particular, with 90% cross-cluster transactions, Flt-B demonstrates 24% lower throughput and 93% higher latency compared to Crd-B due to its all to all communication phases across multiple clusters of multiple enterprises.

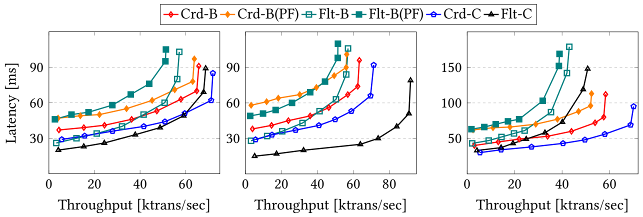

Figure 9: Scalability over spatial domains with (a) 10% intra-shard cross-enterprise, (b) 10% cross-shard intra-enterprise, (c) 10% cross-shard cross-enterprise transactions

5.4 Scalability Across Spatial Domains

In the next set of experiments, we measure the impact of network distance on the performance of the protocols. Clusters of different enterprises are distributed over four different AWS regions, i.e., Tokyo (*TY*), Seoul (*SU*), Virginia (*VA*), and California (*CA*) with the average Round-Trip Time (RTT): *TY* ⇌ *SU*: 33 ms, *TY* ⇌ *VA*: 148 ms, *TY* ⇌ *CA*: 107 ms, *SU* ⇌ *VA*: 175 ms, *SU* ⇌ *CA*: 135 ms, and *VA* ⇌ *CA*: 62 ms.

We consider workloads with 90% *internal* transactions (the typical setting in partitioned databases [82]) and repeat the previous experiments, i.e., intra-shard cross-enterprise transactions (Figure 6(a)), cross-shard intra-enterprise transactions (Figure 7(a)), and cross-shard cross-enterprise transactions (Figure 8(a)).

Since in Fabric, Fabric++, and FastFabric, all transactions are ordered by the same set of orderers, placing endorser nodes that execute transactions of different enterprises and orderer nodes in different locations far from each other is not plausible. Hence, we do not perform this set of experiments for Fabric and its variants.

With 10% intra-shard cross-enterprise transactions (Figure 9(a)), Flt-B demonstrates higher latency due to the message complexity of the protocol that requires all nodes to multicast messages to each other over a wide area network. With 10% cross-shard intra-enterprise transactions (Figure 9(b)), Flt-C demonstrates the best performance because clusters belong to the same enterprise and Qanaat processes transactions using a CFT protocol.

Finally, With 10% cross-shard cross-enterprise transactions (Figure 9(c)), the coordinator-based protocols show better performance because of the two all to all communication phases of the flattened protocols that take place among distant clusters. Compared to the single datacenter setting, the overhead of using the privacy firewall mechanism is also reduced. With 10% intra-shard cross-enterprise transactions, Crd-B(PF) shows 3% lower throughput and 10% higher latency Compared to Crd-B while with the same workload and in single datacenter setting, Crd-B(PF) demonstrates 6% lower throughput and 20% higher latency.

6 RELATED WORK

A *permissioned* blockchain system [2, 12, 30, 44, 45, 51, 58, 64, 72, 75, 78] consists of a set of known, identified but possibly untrusted participants (permissioned blockchains are analyzed in several surveys and empirical studies [7, 26, 29, 37, 38, 74, 79, 79]). Several blockchains support confidential transactions in both the permissioned [28, 63] and permissionless [25, 49] settings. These systems, however, are not scalable and only support simple, financial transfers, not more complex database updates.

Hyperledger Fabric [12] stores confidential data in private data collections [50] replicated on authorized enterprises. The hash of all private transactions, however, is appended to the single global ledger of every node resulting in low performance.

Several variants of Fabric, e.g., Fast Fabric [45], XOX Fabric [44], Fabric++ [78], and FabricSharp [75], have been presented to improve its performance. Such systems, however, do not address the confidential data leakage and consistency challenges of Fabric. Fabric can also be combined with secure multiparty computation [17] but this does not address performance issues.

Caper [5] supports private internal and public global transactions of collaborative enterprises. Caper, however, does not address transactions among a subset of enterprises, consistency across collaboration workflows, confidential data leakage prevention, and multi-shard enterprises.

The zero-knowledge proofs techniques have been used in different blockchain systems such as Hawk [56], Arbitrum [53], Zkay [80], Ekiden [32], and zexe [21] to support confidential transactions. These systems focus on the application layer and do not attempt to address the consensus and data layers that are the focus of Qanaat. Hence, these systems are complementary to Qanaat, and some of these tools could be adapted to provide private contracting capabilities on top of the Qanaat ledger.

Data sharding techniques are used in distributed databases [16, 23, 34, 36, 43, 52, 82, 83] with crash-only nodes and in permissioned blockchain systems, e.g., AHL [35], Blockplane [65], chainspace [3], SharPer [8], and Saguaro [10] with Byzantine nodes. Qanaat, in contrast to all these systems, supports multi-enterprise environments and guarantees confidentiality.

7 CONCLUSION

In this paper, we proposed Qanaat, a permissioned blockchain system to support the scalability and confidentiality requirements of multi-enterprise applications. To guarantee collaboration confidentiality, Qanaat constructs a hierarchical data model consisting of a set of data collections for each collaboration workflow. Every subset of enterprises is able to form a confidential collaboration private from other enterprises and execute transactions on a private data collection shared between only the involved enterprises. To prevent confidential data leakage, Qanaat utilizes the privacy firewall mechanism [86]. To support scalability, each enterprise partitions its data into different data shards. Qanaat further presents a transaction ordering scheme that enforces only the necessary and sufficient constraints to guarantee data consistency. Finally, a suite of consensus protocols is presented to process different types of intra-shard and cross-shard transactions within and across enterprises. Our experimental results clearly demonstrate the scalability of Qanaat compared to Fabric and its variants. Moreover, while coordinator-based protocols demonstrate better performance in cross-enterprise transactions, flattened protocols show higher performance in cross-shard transactions.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful feedback and suggestions. This work is funded by NSF grants CNS-1703560, and CNS-2104882 and by ONR grant N00014-18-1-2021.

REFERENCES

[1] Abdullahi Tunde Aborode, Wireko Andrew Awuah, Suprateeka Talukder, Ajagbe Abayomi Oyeyemi, Esther Patience Nansubuga, Paciencia Machai, Heather Tillewein, and Christian Inya Oko. 2022. Fake COVID-19 vaccinations in Africa. *Postgraduate medical journal* 98, 1159 (2022), 317–318.

[2] Rishav Raj Agarwal, Dhruv Kumar, Lukasz Golab, and Srinivasan Keshav. 2020. Consentio: Managing consent to data access using permissioned blockchains. In *Int. Conf. on Blockchain and Cryptocurrency (ICBC)*. IEEE, 1–9.

[3] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. 2018. Chainspace: A sharded smart contracts platform. In *Network and Distributed System Security Symposium (NDSS)*.

[4] Shahriar Tanvir Alam, Sayem Ahmed, Syed Mithun Ali, Sudipa Sarker, Golam Kabir, et al. 2021. Challenges to COVID-19 vaccine supply chain: Implications for sustainable development goals. *International Journal of Production Economics* 239 (2021), 108193.

[5] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. CAPER: a cross-application permissioned blockchain. *Proc. of the VLDB Endowment* 12, 11 (2019), 1385–1398.

[6] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. ParBlockchain: Leveraging Transaction Parallelism in Permissioned Blockchain Systems. In *Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, 1337–1347.

[7] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2021. Permissioned Blockchains: Properties, Techniques and Applications. In *SIGMOD Int. Conf. on Management of Data*. 2813–2820.

[8] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2021. SharPer: Sharding Permissioned Blockchains Over Network Clusters. In *SIGMOD Int. Conf. on Management of Data*. ACM, 76–88.

[9] Mohammad Javad Amiri, Joris Duguépéroux, Tristan Allard, Divyakant Agrawal, and Amr El Abbadi. 2021. SEPAR: Separ: Towards Regulating Future of Work Multi-Platform Crowdworking Environments with Privacy Guarantees. In *Proceedings of The Web Conf. (WWW)*. 1891–1903.

[10] Mohammad Javad Amiri, Ziliang Lai, Liana Patel, Boon Thau Loo, Eric Loo, and Wenchao Zhou. 2021. Saguaro: Efficient Processing of Transactions in Wide Area Networks using a Hierarchical Permissioned Blockchain. *arXiv preprint arXiv:2101.08819* (2021).

[11] Mohammad Javad Amiri, Boon Thau Loo, Divyakant Agrawal, and Amr El Abbadi. 2021. Qanaat: A Scalable Multi-Enterprise Permissioned Blockchain System with Confidentiality Guarantees. *arXiv preprint arXiv:2107.10836* (2021).

[12] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, et al. 2018. Hyperledger Fabric: a distributed operating system for permissioned blockchains. In *European Conf. on Computer Systems (EuroSys)*. ACM, 30.

[13] Elli Androulaki, Christian Cachin, Angelo De Caro, and Eleftherios Kokoris-Kogias. 2018. Channels: Horizontal scaling and confidentiality on permissioned blockchains. In *European Symposium on Research in Computer Security (ESORICS)*. Springer, 111–131.

[14] David W Archer, Dan Bogdanov, Yehuda Lindell, Liina Kamm, Kurt Nielsen, Jakob Illeborg Pagter, Nigel P Smart, and Rebecca N Wright. 2018. From keys to databases—real-world applications of secure multi-party computation. *Comput. J.* 61, 12 (2018), 1749–1771.

[15] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. 2016. Medrec: Using blockchain for medical data access and permission management. In *Int. Conf. on Open and Big Data (OBD)*. IEEE, 25–30.

[16] Jason Baker, Chris Bond, James C Corbett, JJ Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, and Vadim Yushprakh. 2011. Megastore: Providing scalable, highly available storage for interactive services. In *Conf. on Innovative Data Systems Research (CIDR)*.

[17] Fabrice Benhamouda, Shai Halevi, and Tzipora Halevi. 2019. Supporting private data on hyperledger fabric with secure multiparty computation. *IBM Journal of Research and Development* 63, 2/3 (2019), 3–1.

[18] Gianluca Benigno, Julian di Giovanni, Jan J.J. Groen, and Adam Noble. 2022. Global Supply Chain Pressure Index: May 2022 Update. https://libertystreeteconomics.newyorkfed.org/2022/05/global-supply-chain-pressure-index-may-2022-update/.

[19] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. 2013. DepSky: dependable and secure storage in a cloud-of-clouds. *Transactions on Storage (TOS)* 9, 4 (2013), 12.

[20] Alysson Neves Bessani, Eduardo Pelison Alchieri, Miguel Correia, and Joni Silva Fraga. 2008. DepSpace: a Byzantine fault-tolerant coordination service. In *ACM SIGOPS/EuroSys European Conference on Computer Systems*. 163–176.

[21] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. 2020. Zexe: Enabling decentralized private computation. In *Symposium on Security and Privacy (SP)*. IEEE, 947–964.

[22] Gabriel Bracha and Sam Toueg. 1985. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)* 32, 4 (1985), 824–840.

[23] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, et al. 2013. TAO: Facebook's Distributed Data Store for the Social Graph. In *Annual Technical Conf. (ATC)*. USENIX Association, 49–60.

[24] Lucien Bruggeman and Sasha Pezenik. 2022. Emergent BioSolutions discarded ingredients for 400 million COVID-19 vaccines, probe finds. https://abcnews.go.com/US/emergent-biosolutions-discarded-ingredients-400-million-covid-19/story?id=84604285.

[25] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2020. Zether: Towards privacy in a smart contract world. In *Int. Conf. on Financial Cryptography and Data Security*. Springer, 423–443.

[26] Christian Cachin and Marko Vukolić. 2017. Blockchain Consensus Protocols in the Wild. In *Int. Symposium on Distributed Computing (DISC)*. 1–16.

[27] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *Symposium on Operating systems design and implementation (OSDI)*, Vol. 99. USENIX Association, 173–186.

[28] Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed Kosba, Ari Juels, and Elaine Shi. 2017. Solidus: Confidential distributed ledger transactions via PVORM. In *SIGSAC Conf. on Computer and Communications Security*. ACM, 701–717.

[29] Jeeta Ann Chacko, Ruben Mayer, and Hans-Arno Jacobsen. 2021. Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric. In *SIGMOD Int. Conf. on Management of Data*. ACM, 221–234.

[30] JP Morgan Chase. 2016. Quorum white paper.

[31] Jingjing Chen, Tiefeng Cai, Wenxiu He, Lei Chen, Gang Zhao, Weiwen Zou, and Lingling Guo. 2020. A blockchain-driven supply chain finance application for auto retail industry. *Entropy* 22, 1 (2020), 95.

[32] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2019. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *European Symposium on Security and Privacy (EuroS&P)*. IEEE, 185–200.

[33] Om Prakash Choudhary, Priyanka, Indraj Singh, Teroj A Mohammed, and Alfonso J Rodriguez-Morales. 2021. Fake COVID-19 vaccines: scams hampering the vaccination drive in India and possibly other countries. *Human Vaccines & Immunotherapeutics* (2021), 1–2.

[34] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, et al. 2013. Spanner: Google's globally distributed database. *Transactions on Computer Systems (TOCS)* 31, 3 (2013), 8.

[35] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards Scaling Blockchain Systems via Sharding. In *SIGMOD Int. Conf. on Management of Data*. ACM.

[36] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: amazon's highly available key-value store. In *Operating Systems Review (OSR)*, Vol. 41. ACM SIGOPS, 205–220.

[37] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Ji Wang. 2018. Untangling blockchain: A data processing view of blockchain systems. *IEEE transactions on knowledge and data engineering* 30, 7 (2018), 1366–1385.

[38] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. Blockbench: A framework for analyzing private blockchains. In *SIGMOD Int. Conf. on Management of Data*. ACM, 1085–1100.

[39] Sisi Duan and Haibin Zhang. 2016. Practical state machine replication with confidentiality. In *Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 187–196.

[40] David Evans, Vladimir Kolesnikov, and Mike Rosulek. 2017. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security* 2, 2-3 (2017).

[41] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)* 32, 2 (1985), 374–382.

[42] Ariel Gabizon and Zachary J Williamson. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. IACR ePrint 2019/953. (2019).

[43] Lisa Glendenning, Ivan Beschastnikh, Arvind Krishnamurthy, and Thomas Anderson. 2011. Scalable consistency in Scatter. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 15–28.

[44] Christian Gorenflo, Lukasz Golab, and Srinivasan Keshav. 2020. XOX Fabric: A hybrid approach to transaction execution. In *Int. Conf. on Blockchain and Cryptocurrency (ICBC)*. IEEE, 1–9.

[45] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. 2019. Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second. In *Int. Conf. on Blockchain and Cryptocurrency (ICBC)*. IEEE, 455–463.

[46] Christian Gorenflo, Stephen Lee, Lukasz Golab, and S. Keshav. 2019. FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second. *arXiv preprint arXiv:1901.00910* (2019).

[47] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2020. ResilientDB: Global Scale Resilient Blockchain Fabric. *Proceedings of the VLDB Endowment* 13, 6 (2020), 868–883.

[48] Siyuan Han, Zihuan Xu, Yuxiang Zeng, and Lei Chen. 2019. Fluid: A blockchain based framework for crowdsourcing. In *SIGMOD Int. Conf. on Management of Data*. ACM, 1921–1924.

[49] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2016. Zcash protocol specification. *GitHub: San Francisco, CA, USA* (2016).

[50] Hyperledger. [n.d.]. Private Data Collections: A High-Level Overview. https://www.hyperledger.org/blog/2018/10/23/private-data-collections-a-high-level-overview.

[51] Zsolt István, Alessandro Sorniotti, and Marko Vukolić. 2018. StreamChain: Do Blockchains Need Blocks?. In *Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers (SERIAL)*. ACM, 1–6.

[52] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alexander Rasin, Stanley Zdonik, Evan PC Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, et al. 2008. H-store: a high-performance, distributed main memory transaction processing system. *Proc. of the VLDB Endowment* 1, 2 (2008), 1496–1499.

[53] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. 2018. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium)*. 1353–1370.

[54] Maher Khan and Amy Babay. 2021. Toward Intrusion Tolerance as a Service: Confidentiality in Partially Cloud-Based BFT Systems. In *Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE, 14–25.

[55] K. Korpela, J. Hallikas, and T. Dahlberg. 2017. Digital supply chain transformation toward blockchain integration. In *Hawaii Int. Conf. on system sciences (HICSS)*.

[56] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Symposium on security and privacy (SP)*. IEEE, 839–858.

[57] Ahmed Kosba, Charalampos Papamanthou, and Elaine Shi. 2018. xJsnark: A framework for efficient verifiable computation. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 944–961.

[58] Jae Kwon. 2014. Tendermint: Consensus without mining. (2014).

[59] Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* 21, 7 (1978), 558–565.

[60] Leslie Lamport. 2001. Paxos made simple. *ACM Sigact News* 32, 4 (2001), 18–25.

[61] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. 2019. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2111–2128.

[62] Michael A Marsh and Fred B Schneider. 2004. CODEX: A robust and secure secret distribution system. *Transactions on Dependable and secure Computing* 1, 1 (2004), 34–47.

[63] Neha Narula, Willy Vasquez, and Madars Virza. 2018. zkledger: Privacy-preserving auditing for distributed ledgers. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. 65–80.

[64] Senthil Nathan, Chander Govindarajan, Adarsh Saraf, Manish Sethi, and Praveen Jayachandran. 2019. Blockchain meets database: design and implementation of a blockchain relational database. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1539–1552.

[65] Faisal Nawab and Mohammad Sadoghi. 2019. Blockplane: A global-scale byzantizing middleware. In *2019 IEEE 35th Int. Conf. on Data Engineering (ICDE)*. IEEE, 124–135.

[66] Diego Ongaro and John K Ousterhout. 2014. In search of an understandable consensus algorithm. In *Annual Technical Conf. (ATC)*. USENIX Association, 305–319.

[67] World Health Organization. 2021. Medical Product Alert N°5/2021: Falsified COVISHIELD vaccine. https://www.who.int/news/item/31-08-2021-medical-product-alert-n-5-2021-falsified-covishield-vaccine.

[68] Ricardo Padilha and Fernando Pedone. 2011. Belisarius: BFT storage with confidentiality. In *2011 IEEE 10th International Symposium on Network Computing and Applications*. IEEE, 9–16.

[69] Dan Patterson. 2021. Hackers are attacking the COVID-19 vaccine supply chain. https://www.cbsnews.com/news/covid-19-vaccine-hackers-supply-chain/.

[70] Zhe Peng, Cheng Xu, Haixin Wang, Jinbin Huang, Jianliang Xu, and Xiaowen Chu. 2021. P2B-Trace: Privacy-Preserving Blockchain-based Contact Tracing to Combat Pandemics. In *SIGMOD Int. Conf. on Management of Data*. 2389–2393.

[71] Zhe Peng, Jianliang Xu, Xiaowen Chu, Shang Gao, Yuan Yao, Rong Gu, and Yuzhe Tang. 2021. Vfchain: Enabling verifiable and auditable federated learning via blockchain systems. *IEEE Transactions on Network Science and Engineering* (2021).

[72] Ji Qi, Xusheng Chen, Yunpeng Jiang, Jianyu Jiang, Tianxiang Shen, Shixiong Zhao, Sen Wang, Gong Zhang, Li Chen, Man Ho Au, et al. 2021. Bidl: A High-throughput, Low-latency Permissioned Blockchain Framework for Datacenter Networks. In *Symposium on Operating Systems Principles (SOSP)*. ACM SIGOPS, 18–34.

[73] Steve Reilly, Jason Paladino, Jonathan Lambert, and Matt Stiles. 2022. Fake vaccine cards are everywhere. It's a public health nightmare. https://www.grid.news/story/science/2022/01/25/fake-vaccine-cards-are-everywhere-its-a-public-health-nightmare/.

[74] Pingcheng Ruan, Tien Tuan Anh Dinh, Dumitrel Loghin, Meihui Zhang, Gang Chen, Qian Lin, and Beng Chin Ooi. 2021. Blockchains vs. Distributed Databases: Dichotomy and Fusion. In *SIGMOD Int. Conf. on Management of Data*. 1504–1517.

[75] Pingcheng Ruan, Dumitrel Loghin, Quang-Trung Ta, Meihui Zhang, Gang Chen, and Beng Chin Ooi. 2020. A Transactional Perspective on Execute-order-validate Blockchains. In *SIGMOD Int. Conf. on Management of Data*. ACM, 543–557.

[76] Sarah Schiffling and Nikolaos Valantasis Kanellos. 2022. Shanghai: world's biggest port is returning to normal, but supply chains will get worse before they get better. https://theconversation.com/shanghai-worlds-biggest-port-is-returning-to-normal-but-supply-chains-will-get-worse/before-they-get-better-182720.

[77] Fred B Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *Computing Surveys (CSUR)* 22, 4 (1990), 299–319.

[78] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. 2019. Blurring the lines between blockchains and database systems: the case of hyperledger fabric. In *SIGMOD Int. Conf. on Management of Data*. ACM, 105–122.

[79] Man-Kit Sit, Manuel Bravo, and Zsolt István. 2021. An experimental framework for improving the performance of BFT consensus for future permissioned blockchains. In *Proceedings of the 15th ACM Int. Conf. on Distributed and Event-based Systems*. 55–65.

[80] Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin Vechev. 2019. zkay: Specifying and enforcing data privacy in smart contracts. In *ACM SIGSAC Conf. on Computer and Communications Security (CCS)*. 1759–1776.

[81] Judy Stone. 2021. How Counterfeit Covid-19 Vaccines And Vaccination Cards Endanger Us All. https://www.forbes.com/sites/judystone/2021/03/31/how-counterfeit-covid-19-vaccines-and-vaccination-cards-endanger-us-all/?sh=eaddb0e36495.

[82] Rebecca Taft, Essam Mansour, Marco Serafini, Jennie Duggan, Aaron J Elmore, Ashraf Aboulnaga, Andrew Pavlo, and Michael Stonebraker. 2014. E-store: Fine-grained elastic partitioning for distributed transaction processing systems. *Proc. of the VLDB Endowment* 8, 3 (2014), 245–256.

[83] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J Abadi. 2012. Calvin: fast distributed transactions for partitioned database systems. In *SIGMOD Int. Conf. on Management of Data*. ACM, 1–12.

[84] Feng Tian. 2017. A supply chain traceability system for food safety based on HACCP, blockchain & Internet of things. In *Int. Conf. on service systems and service management (ICSSSM)*. IEEE, 1–6.

[85] Robin Vassantlal, Eduardo Alchieri, Bernardo Ferreira, and Alysson Bessani. 2022. COBRA: Dynamic Proactive Secret Sharing for Confidential BFT Services. In *Symposium on Security and Privacy (SP)*. IEEE, 1528–1528.

[86] Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi, and Mike Dahlin. 2003. Separating agreement from execution for byzantine fault tolerant services. *Operating Systems Review (OSR)* 37, 5 (2003), 253–267.