Soteria: A Provably Compliant User Right Manager Using a Novel Two-Layer Blockchain Technology

Wei-Kang Fu[3] Yi-Shan Lin[1] Giovanni Campagna[2]
Chun-Ting Liu[1] De-Yi Tsai[3] Chung-Huan Mei[1]
Edward Y. Chang[1,2] Shih-Wei Liao[3] Monica S. Lam[2]
HTC DeepQ[1] Stanford University[2] National Taiwan University[3]
echang@cs.stanford.edu lam@cs.stanford.edu

***Abstract*—Soteria is a user right management system designed to safeguard user-data privacy in a transparent and provable manner in compliance to regulations such as GDPR and CCPA. Soteria represents user data rights as formal executable sharing agreements, which can automatically be translated into a human readable form and enforced as data are queried. To support revocation and to prove compliance, an indelible, audited trail of the hash of data access and sharing agreements are stored on a two-layer distributed ledger. The main chain ensures partition tolerance and availability (PA) properties while side chains ensure consistency and availability (CA), thus providing the three properties of the CAP (consistency, availability, and partition tolerance) theorem. Besides depicting the two-layer architecture of Soteria, this paper evaluates representative consensus protocols and recommends side-chain and inter-chain management strategies for improving latency and throughput.**

***Index Terms*—blockchain, privacy, ccpa, gdpr**

I. Introduction

Artificial intelligence (AI) has the potential to improve the quality of many application domains. In order to effectively train AI models, an application typically requires large quantities of personal information (PI) from users. To address data privacy issues, regulations such as GDPR [1] and CCPA [4] in the general domain and HIPAA [2] in the medical domain need to be upheld. Businesses are required to comply with the *consumer rights* in a *provable* way. The eight consumer rights of GDPR and the six of CCPA can be summarized into three categories: 1) right to *consent*, rectify, and delete; 2) right to be *informed*; and 3) right to *access* and transfer.

- Consent: users must explicitly opt-in and always have the ability to opt-out of PI collection.
- Informed: users have the right to know how their PI is collected, used, and shared.
- Access: users have the right to access his/her own data, transfer data to another person or entity, and erase data.

To protect privacy rights in a provable manner, we propose Soteria, a user-right management system with a distributed ledger platform. Soteria provides a formal end-to-end solution that automatically maps user agreements to share data in natural language into formal compliance code. Our *executable sharing agreements* (ESA) are a formal representation of sharing agreements that can specify a superset of the rights protected under GDPR and CCPA. These agreements are translated into formal Satisfiability Modulo Theory (SMT) formulas for enforcement. Agreements can be translated back into natural language automatically so users can review and audit. (Most users may not be able to understand an ESA written in SMT formulas.) For provable compliance to privacy regulations, Soteria uses a distributed ledger to support auditability and revocability. We create an indelible trail of records by first logging every agreement signed and every query made, and putting a hash of the log on a distributed ledger. Soteria ensures all transactions (including permission, revocation, data access, and deletion) on PI leave indelible and consistent records for public audit. Provability is essential in the court of law to resolve disputes.

Soteria's ledger system employs a two-layer blockchain architecture to allow all three CAP (consistency, availability, and partition tolerance) properties [9] to be simultaneously satisfied. The base-layer blockchain guarantees PA (partition tolerance and availability) properties to ensure transparency, whereas the side chains guarantee CA (consistency and availability) properties to ensures provability. Soteria ensures performance scalability in both latency and throughput. To this end, Soteria uses permissioned side chains to achieve low latency, and replicates side chains when necessary to achieve high throughput. At the same time, the main chain uses a decentralized and permissionless blockchain to ensure transparency for public auditing. We detail our protocol selections for the base blockchain and side chains, and inter-chain management policies in Section III.

The rest of this paper is organized into four sections. Section II depicts Soteria's architecture and its user right management system. Section III presents and evaluates consensus protocols for both the main chain and side chains qualitatively. In Section IV, we report our quantitative performance evaluation on readily deployable protocols and suggest main-chain, side-chain, and inter-chain management strategies. We offer our concluding remarks and discuss future work in Section V.

II. Architecture of Soteria

We use the terms *user* and *consumer* interchangeably to refer to the owner of data. (Data consumer is the term defined

IEEE Infrastructure Conference, San Francisco, October, 2020.

in CCPA to refer to an application user whose data the regulation aims to protect.) We use *business* and *company* to refer to the collector and custodian of user/consumer data. A user, and a business with that user's consent, can access the data collected from the user stored at the business.

A. Components and Design Goals

Figure 1 presents the components of Soteria. Soteria consists of three modules: in additional to its distributed ledger (DLT) that employs a two-layer blockchain, URM is user-right management for and ATS is for audit trail service .

- *User right management* (URM): URM stores descriptions of user data and metadata that are collected and stored by a company.
- *Distributed ledger* (DTL): DTL is our two-layer blockchain that satisfies Soteria's requirements including privacy, throughput and latency.
- *Audit trail service* (ATS): ATS stores incurred transactions on data items for transparent auditing.

Soteria is designed to address the following challenges:

1) Today users are asked to consent to long hard-to-read agreements. How can we write agreements that can be understandable to users?
2) How do we ensure that the agreements users sign translate into a faithful implementation?
3) How can a company prove that it is compliant? In particular, how do consumers ensure that no accesses to revoked data are performed?

While Section III depicts the ledger design in DTL for addressing the provability requirement, the remainder of this section presents how URM complies with regulations of consent, informed, and access described in Section I.

B. ESA: Executable Sharing Agreement

In existing systems, users are required to agree to long documents that are not understandable. Furthermore, because these agreements are expressed in natural language, which can be ambiguous, it is not clear what the effect of the agreements, or whether a certain company is truly in compliance. Instead, we propose *executable sharing agreement* (ESA), which have a well-defined unambiguous semantics; that is, whether a specific transaction complies with the agreement can be verified automatically. Furthermore, ESAs can automatically be converted into natural language unambiguously. This ensures that the contract is understandable to users and auditors.

Our ESA notation is inspired by the previously proposed ThingTalk language [13], designed originally for personal data sharing [15]. The syntax of an ESA is as follows:

$$\gamma, \pi(r,p) : [f_1, f_2, \dots] \ \texttt{of} \ d, \pi(f_1, f_2, \dots)$$

This statement reads as follows: "for consumer γ, the fields $f_1, f_2, \dots$ from domain d can be shared with any *requester* r for purpose p, provided that the predicates $\pi(r,p)$ and $\pi(f_1, f_2, \dots)$ are satisfied.

So, for example, to express that they are willing to share their abnormal PSA (a protein called prostate-specific antigen) with Stanford Medical Center, including their age and ethnicity but not their name, and only for research purposes, a patient named "Bob" would issue an ESA agreement of the form:

$$\gamma = \text{"Bob"}, r = \text{"Stanford Medical Center"} \wedge p = \text{research} : [age, ethnicity, PSA] \ \texttt{of} \ \text{EHR}, PSA \geq 2$$

1) Translating to and from Natural Language: As consumers are not expected to understand formal languages, the ESA notation is designed to provide a natural language interface. The sharing agreements can be translated from formal to natural language using a rule-based translation. The previous example can be expressed in natural language as:

> "Stanford Medical Center can read the age, ethnicity and PSA of Bob's EHR for research and if the PSA is greater than or equal to 2."

While the automatically generated sentences can be verbose and clunky due to the rule-based translation, they are understandable, and they are guaranteed to correspond exactly to the code of the agreement. Furthermore, the ESA notation is designed so that a user can define their own sharing agreement in natural language. Previous work [13] has shown that it is possible to automatically translate natural language access controls into attribute-based policies for personal data sharing [14], [15], and the same semantic parsing technology is used here.

C. ESA Enforcement

All writes to and reads from the database containing user data must go through the Soteria interface to ensure compliance to all the sharing agreements, which represent user consent. Soteria automatically includes an *owner* field for each row of the database. Every database access is rewritten to include the sharing agreements constraints; it is timestamped and logged for later auditing.

1) Verification of SQL Queries for Auditing: To prove compliance to an external auditor, Soteria stores the requester, the purpose and the final query, right before it is issued to the database, including all the clauses related to the sharing agreements. Using the set of the sharing agreements in force at the time, the auditor can then formally verify that the query was compliant. Given a query from requester r for purpose p in the audit logs of the form:

$$\texttt{SELECT} \ \bar{f} \ \texttt{FROM} \ t \ \texttt{WHERE} \ \pi$$

and sharing agreements of the form:

$$\gamma_1, \pi_1(r,p) : [f_1, f_2, \dots] \ \texttt{of} \ t, \pi_1(f_1, f_2, \dots)$$
$$\gamma_2, \pi_2(r,p) : [f_1, f_2, \dots] \ \texttt{of} \ t, \pi_2(f_1, f_2, \dots)$$
$$\dots$$

the query is compliant if and only if

$$\pi \models (\gamma = \gamma_1 \wedge \pi_1(r,p) \wedge \pi_1(f_1, f_2, \dots)) \vee (\gamma = \gamma_2 \wedge \pi_2(r,p) \wedge \pi_2(f_1, f_2, \dots)) \vee \dots$$

where γ is the consumer who owns a specific row in the database. This logical formula can be verified efficiently using a satisfiability modulo theory (SMT) solver [8].

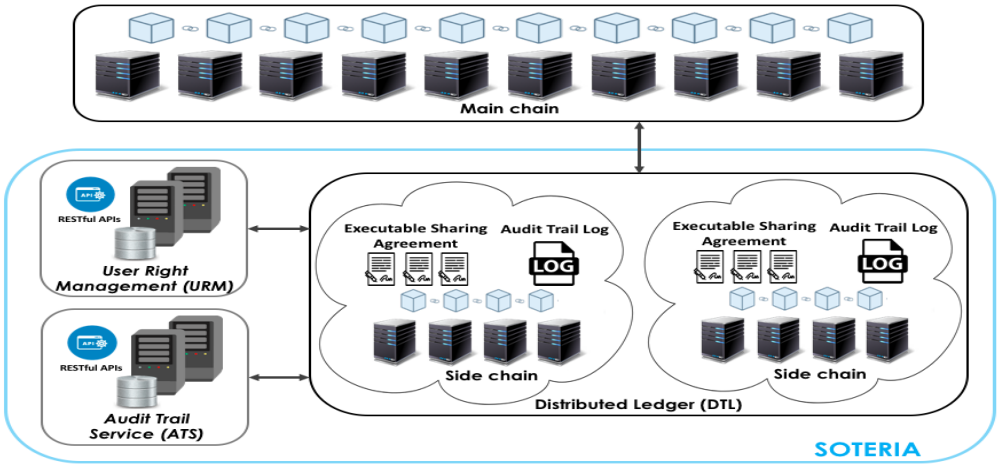


Fig. 1: Soteria Components: URM, DTL, and ATS.

2) Formally Verified SQL Query: To ensure that all queries are compliant, Soteria uses the following algorithm to construct a query that is guaranteed by construction to satisfy the sharing agreements. Given a SQL query from requester r for purpose p of the form:

$$\texttt{SELECT } \bar{f} \texttt{ FROM } t \texttt{ WHERE } \pi$$

and sharing agreements of the form:

$$\begin{aligned} &\gamma_1, \pi_1(r,p) : [f_1, f_2, \ldots] \quad \texttt{of} \quad t, \pi_1(f_1, f_2, \ldots) \\ &\gamma_2, \pi_2(r,p) : [f_1, f_2, \ldots] \quad \texttt{of} \quad t, \pi_2(f_1, f_2, \ldots) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \ldots \end{aligned}$$

Soteria constructs a query of the form:

$$\begin{aligned} &\texttt{SELECT } \bar{f} \cap \{f_1, f_2, \ldots\} \texttt{ FROM } t \texttt{ WHERE } \pi \texttt{ AND} \\ &\quad (\gamma = \gamma_1 \texttt{ AND } \pi_1(r,p) \texttt{ AND } \pi_1(f_1, f_2, \ldots)) \texttt{ OR} \\ &\quad (\gamma = \gamma_2 \texttt{ AND } \pi_2(r,p) \texttt{ AND} \pi_2(f_1, f_2, \ldots)) \texttt{ OR} \ldots, \end{aligned}$$

where γ in the table contains the owner of that row.

It is possible to prove that the result set of this query is consistent with the sharing agreement. Only the fields allowed by the agreement are returned, and a row is included only if the predicates in the sharing agreement in force for the row owner are satisfied.

3) Enforcing Requester and Purpose: A data requester can be an app user defined in CCPA or a company that stores user data. The URM component of Soteria assumes that the identity of the data requester has been verified by the data transport protocol. Soteria does not mandate any specific transport protocol; meaningful protocols could be REST, messaging-based [15], or could even be through physical media.

Furthermore, Soteria does not verify the purpose of the data access. The design assumes that at identification time, the data transport protocol will also compute the allowed purpose of data access. For example, using REST data transport, a medical data requester could be issued two different access tokens, one for clinical purposes and the other for research purposes. The correct purpose can be established through an existing business agreement between the data provider and data requester. If the data requester then uses the data in a manner inconsistent with the agreed purpose, the provider can use Soteria to establish that the fault lies with the requester.

4) Enforcing Access on Write: To limit the data stored by a business entity such as AWS, queries that manipulate data (insert, update, delete) are also intercepted by Soteria, and modified to match the ESA governing the storage of data. Write queries not allowed by any ESA are not executed and not logged.

To support permanent deletion of the data upon request by the consumer, the data itself (included in the `VALUES` and `SET` clauses of the insert and update queries) is masked in the audit log, as the audit log is immutable and append-only. For this reason, an ESA limiting writes that depend on the specific values cannot be audited after the fact. Soteria disallows such ESA: only the set of fields to write can be controlled.

Deletions are a special case of write, because they reduce the data that is stored about a specific customer rather than store new data. Hence, deletion is always allowed regardless of the current ESA. All deletions are logged, to prove that they were executed properly.

Additionally, upon revoking an ESA that limits the allowed writes to the database, some columns that were previously allowed might becomes disallowed. In that case, Soteria overwrites the data to set those columns to `NULL`. In case all columns are now disallowed, such as when all ESAs for a customer are revoked, Soteria deletes the row entirely. Both the database write (update or delete) and the ESA revocation are logged for later auditing.

D. ESA Storage and Audit

To support long-term auditability, as well as revocation of contracts, Soteria makes use of a distributed ledger (blockchain) to track which sharing agreements are in force. The use of the blockchain provides a global ordering of all

the events across all parties in the system; the events include issuing a sharing agreement, accessing the data, and revoking a sharing agreement. This global ordering ensures it is always possible to verify whether a sharing agreement was in force when a data transaction occurred, without disputes.

Sharing agreements can potentially be privacy-sensitive themselves; for example, the sharing agreement in the previous section can reveal that the patient is likely to be male. For this reason, Soteria only stores the hash of the sharing agreement code, and the hash of each transaction, in the public blockchain. Upon request by a competent authority (e.g. under subpoena in a civil dispute), a data provider using Soteria can reveal the full audit logs, including the full code of the sharing agreements, and the exact transactions performed. These logs can then be matched to the hashes stored in the public blockchain to verify that they were not tampered with.

Soteria includes three types of events in the blockchain:

1) *Sharing Agreement Deployment* (ESAD). An ESAD event occurs when a new sharing agreement is created between a user and a business. An ESAD transaction on the blockchain (ESAD-TX) includes both user and business addresses, the hash of the agreement code, ESA deployment date, and ESA validity status (set to *true*).
2) *Data Transaction* (DATA-TX). A data transaction indicates the transfer of data between a data provider and a data requester. DATA-TX records in the blockchain include the address of the data provider, the address of the data requester, and the hash of the exact query executed against the database. Note that, as described in the previous section, the exact query will include a reference to the consumers and their sharing constraints. Hence, given the raw audit logs, paired with the hash in the blockchain, it is possible to verify that the query was valid when executed.
3) *Sharing Agreement Revocation* (ESAR). A sharing agreement can be revoked by a user, making it invalid. Since the ledger is append-only, a revocation is implemented by creating a new sharing agreement transaction with the validity status set to *false*. Note that a rectification request issued by a user is executed in two consecutive transactions, an ESAR to revoke an existing consent and then an ESAD to create a new consent.

III. DLT Consensus Protocols

Soteria adopts blockchain technology to maintain its distributed ledger among several institutions and users. To achieve high throughput and low latency, Soteria uses a two-layer blockchain (we previously proposed in [26]). The base layer is decentralized similar to Ethereum, and its side chains are entrusted by a group of selected validators, known as *juries*. Soteria may spawn multiple independently operated side chains for improving overall throughput. An ESA'a agreement, access, and revocation are all logged on the same side chain to ensure a local consistent order on that side chain.

Recall that the CAP theorem [9] states that a distributed database system can only satisfy two of the three properties: consistency, availability, and partition tolerance. While the base-layer chain ensures availability and partition tolerance, the side chains ensure consistency and availability. Using a permissioned side chain with a jury pool, Soteria can improve both throughput and latency by limiting the number of juries. Note that public audit is performed on the permissionless main chain. There is a delay between when a transaction is committed on a side chain and when the hash of the side-chain block containing the transaction is written onto the main chain. A user can be informed that her ESA signed with the business has been recorded on a side chain. However, only after the main chain has been updated, the user (or through the user's agent) can verify that the contract has indeed been signed and was not altered. (Inter-chain latency will be discussed in greater details shortly.)

Soteria's ledger requires a consensus protocol since it supports contract revocation. A consensus protocol ensures that all validators agree on a unique order of transactions on the ledger. Note that without this strict access permission and access revocation order requirement, an append-only log suffices to support the decentralized auditability requirement.

There are several types of consensus protocols, each of which enjoys some pros and suffers some cons. An application selects a particular consensus protocol for its desired performance objectives (e.g., latency, throughput, and consistency). For instance, a protocol that guarantees immediate consistency may trade latency and throughput for achieving the consistency objective. A protocol that requires just weak consistency or eventual consistency can achieve shorter latency and higher throughput[1].

Specifically, the PoX (proof-of-something) family protocols [3], [12], [22], [31] such as Proof-of-Work (PoW), Proof-of-Stake (PoS), and Proof-of-Importance (PoI) offer timely consistency with good network scalability but suffer from high latency and low transaction throughput. On the contrary, the BFT (Byzantine Fault Tolerance) family protocols offer good performance with limited scalability with respect to the number of validators. PoX is thus more suitable for permissionless blockchains (Soteria 's main chain), whereas BFT is more suitable for permissioned blockchains (side chains).

In the remainder of this section, we survey representative BFT consensus protocols including Tendermint [11], Hashgraph [7], HotStuff [36], and Stellar[28]. We summarize in Section III-C a qualitative comparison before presenting our empirical study in Section IV.

A. Blockchain Overview and Terminologies

A blockchain of height H is composed of a sequence of H blocks. Each block h, where $h = 0, \ldots, H-1$, in a blockchain

[1]All consensus protocols discussed in this paper, in either the family of PoX or BFT/PBFT, observe properties including consistency, safety, liveness, and fault tolerance to an extent. A protocol may prioritize e.g., liveness over consistency (accepting eventual consistency instead of enforcing strong consistency). Which properties enjoy higher priorities and the choice of a protocol may boil down to performance and cost consideration. We assume each protocol works correctly as it specifies and claims. Detailed discussion on the properties observed by individual protocols is beyond the scope of this paper.

	Tendermint	Stellar	HotStuff	Hashgraph
Timing Model	Partial Synchronous	Partial Synchronous	Partial Synchronous	Asynchronous
Key Design Goals	Single mode mechanism	Flexible trust	1. Linearity 2. Optimistic responsiveness	High throughput
Fault Tolerance	$\leq \frac{2}{3}$ Voting Power	not available (na)	$\leq \frac{2}{3}$ Voting Power	$\leq \frac{2}{3}$ Voting Power
Message Complexity	$\mathcal{O}(N^2)$ NA $\mathcal{O}(N^3)$	na	$\mathcal{O}(N)$ NA $\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
Scalability	100-1,000	na	≥ 100	$\geq 1{,}000$
Validator Bound	64	43	128	128
Throughput (tx/s)	$4k$	na	$10k$	$\geq 50k$
Latency (s)	5	1.3	10	≥ 10
Hardware Config.	AWS t2.medium	AWS c5d.9xlarge	AWS c5.4xlarge	AWS m4.4xlarge

TABLE I: Consensus Protocols Comparative Analysis. Data Sources: [7], [10], [27], [36]. N denotes # of validators.

contains various numbers of transactions organized into a Merkle tree. The Merkle tree root of the h^{th} block, denoted as R_h, summarizes the information in that block. The h^{th} block, where $h \in [1,\ldots,H-1]$ points to its previous block with pointer B_h. (The first block, or $h = 0$ is the genesis block.) $B_h$is the hash of three components: previous block hash B_{h-1}, the Merkle tree root of current block R_h, and some information from the h^{th} block T_h such as timestamp.

B. Base Chain Protocols

Proof-of-something (PoX) protocols aim to support decentralized permissionless blockchains. As mentioned in Section I, Soteria uses a PoX blockchain as its base chain. The choice of a PoX protocol depends on the factors of cost and *inter-chain consistency latency*, which is defined as the time between when a transaction commits on a side chain and when the root of the transaction's Merkle tree is hashed onto the main chain. Inter-chain consistency is different from transaction consistency. The former ensures a consistent public view of transactions for the purpose of decentralized audits, whereas the latter ensures the validity of individual transactions. Transactions committed on a side chain guarantees a consistent local order on that chain. The main chain, however, does not guarantee total order on all the transactions across all side chains. Once Soteria enforces that a contract revocation must take place on the same side chain where the contract was agreed upon and validated, side chain consistency guarantee suffices.

Access revocation transactions take effect within seconds, as a side chain of Soteria can ensure transaction-commit latency to be within seconds. In other words, once a user revokes a prior permission to access her data, her data will be inaccessible within seconds. For the purpose of auditing, Soteria can only guarantee inter-chain consistency within minutes (inter-chain latency), this suffices for the auditing purpose required by regulations[2]. A PoS (proof-of-stake) protocol such as Ethereum satisfies latency in minutes at a relatively low cost (compared to e.g., Bitcoin). Therefore, Soteria uses Ethereum as its main chain.

[2]E.g., the CCPA announced in January 2020 requires a data holder to respond to an audit request in 45 days.

C. Side Chain Protocols

Byzantine Fault Tolerance (BFT) has a long history with distributed systems [35]. In 1999, [16] implemented "practical" Byzantine Fault Tolerance (PBFT) protocol. PBFT can work in an asynchronous environment, such as the Internet.

PBFT executes in two modes, normal mode and view-change mode. In the normal mode, a leader proposes a candidate value to the other replicas in the pre-prepare phase. PBFT then goes through two successive voting phrases: prepare and commit. If the candidate value is accepted by a replica p_i, as known as a validator, p_i enters the prepare phase and broadcasts a prepare message to others consisting the candidate value. Once p_i collects enough messages, i.e., $2f+1$ messages over $3f+1$ replicas (f denotes the number of Byzantine nodes) and agree on the same value, it enters into the commit phase. In the commit phase, replicas conduct an election similar to the one in the prepare phase to agree that more than $2f$ replicas will write the candidate value into their respective databases. To prevent indefinitely waiting, p_i transitions to view-change if a timeout is triggered. In the view-change mode, replicas start a new view to elect a new leader by sending view-change messages.

The message complexity of BFT/PBFT protocols is between $\mathcal{O}(N^2)$ and $\mathcal{O}(N^3)$. Since N, the size of a jury is typically a small set of trusted validators, latency can be managed[3] to be under 10 seconds.

Of late, there are a large number of BFT/PBFT protocols proposed for prioritizing various performance objectives, e.g., safety, fault tolerance, latency, and throughput (e.g., [5], [18], [19], [21], [24], [25], [17], [30], [32], [34].) A comprehensive survey is beyond the scope of this paper.

For the Soteria' base-chain design, we prioritize reliability under different attacks [29] and preservation of CA properties. We select and evaluate four widely deployed and battle-tested BFT protocols known for their reliability: *Tendermint*, *Stellar*, *HotStuff*, and *Hashgraph*. Due to the page limit, we document the specifications of these four protocols in the Appendix of the extended version of this paper [20]. Table I presents a

[3]Unlike that a decentralized protocols must deal with a potentially massive number of nodes, the administrator of a side chain can adjust N to cap its latency.

qualitative comparison between these four protocols in eight properties:

1) Timing model: timing assumptions of different models including synchronous, asynchronous, and partial synchronous.
2) Design goals: the primary performance goals that a consensus algorithm was designed to achieve. This provides the contextual information for evaluating a protocol. (A protocol designed for achieving low latency should not be derided for its weaknesses in other performance metrics.)
3) Fault tolerance: the upper bound of faulty nodes (or weighted votes) that can cause system failure.
4) Message complexity: the overall message complexity to commit a block. N denotes the number of validators.
5) Scalability: the number of validators that the consensus protocol claims to allow in the consensus process.
6) Validator bound: the largest number of validators that can participate in a consensus protocol.
7) Throughput: the number of transactions per second that can be committed by the majority of validators under the largest number of validators.
8) Latency: the average delay before a transaction is committed under the largest number of validators.

Table I shows only the performance data under the largest number of validators that a protocol can support given its own hardware configuration and assumptions, documented in their own white papers [7], [10], [27], [36]. In Section IV, we presents our own experiments attempting to do a relatively fair comparison.

Note that Stellar is absent in most of the fields in Table I for two reasons. First, fault tolerance and message complexity are highly correlated with the configuration of each validator on the Stellar network. This flexible configuration makes it difficult to analyze Stellar without knowing its network structure. Second, the work of [27] focuses only on reducing latency and assumes that throughput can be improved by adding hardware. Also note that though Table I does not provide an apple-to-apple comparison, it provides a birds-eye view on representative protocols' characteristics.

IV. Experiments

Our experiments were designed to evaluate tradeoffs between latency and throughput under different hardware and software configurations and then devise strategies to improve performance. More specifically, we would like the following questions to be answered:

1) Which protocol(s) can achieve the best performance?
2) What is a cost-effective configuration to support a target throughput and latency performance?
3) Given a required latency, what is the maximum number of validators allowed?
4) What are the side-chain and inter-chain parameters that can be adjusted to achieve desired performance?

Since there is no stable implementation of Hashgraph or Hotstuff[4] to date, we evaluated only the latest versions of Tendermint and Stellar. Note that though we attempt to perform "fair" evaluation on the protocols, it is impossible to achieve absolute fairness due to the fact that engineering quality, parameter settings, and network configurations all can affect experiment results. Therefore, when evaluating the results, we focus on observing trends and patterns to recommend strategies to improve the target performance.

We adopted Tendermint v0.32.1 and Stellar v12.0.0 and deployed them on Google Cloud Platform (GCP). For Tendermint and Stellar, respectively, we created up to 64 validators, and assigned an 8-core Intel(R) Xeon(R) 2.20 GHz CPU with 7.2 GB memory to each validator. Validators were configured into a fully connected topology. To measure realistic network latency, we deployed validators on Google servers in different geographical locations (Taiwan, Singapore, Belgium, and Columbia). Validators were distributed equally to these locations. For example, in the 64-validator implementation, 16 validators were allocated to each of the four geographic locations.

A. Metrics

We define throughput and latency based on an end-user's perspective. Given a set of N validators $V = \{p_1, p_2, \ldots, p_N\}$, throughput denoted as TPS_{avg} is written as

$$\text{TPS}_{\text{avg}} = \frac{\Sigma_{i=1}^{N} \text{TPS}_{\text{avg}}^{i}}{N}, \tag{1}$$

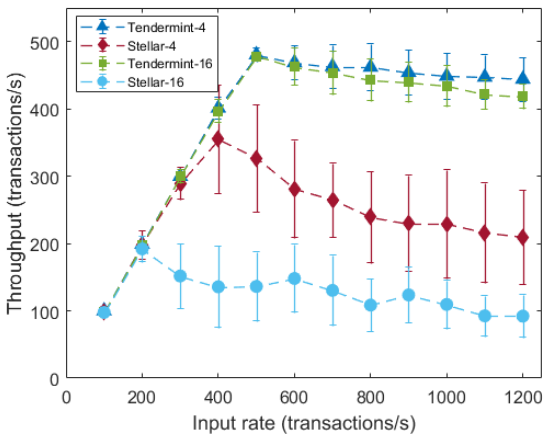
where $\text{TPS}_{\text{avg}}^{i}$ denotes the average throughput of validator p_i, who owns a sequence of blocks $B^i = \{b_0^i, b_1^i, \ldots, b_H^i\}$. In turn, TPS_h^i (throughput of the $h+1^{th}$ block of validator p_i) is the average throughput of all transactions in that block, which is obtained by the number of transactions in the block divided by the commit interval between b_{h-1}^i and b_h^i, which is denoted as Δt_h^i, or

$$\text{TPS}_{\text{avg}}^{i} = \frac{\Sigma_{h=0}^{H_i} \text{TPS}_h^i}{H_i}, \text{ where } \text{TPS}_h^i = \frac{|T_h^i|}{\Delta t_h^i}.$$

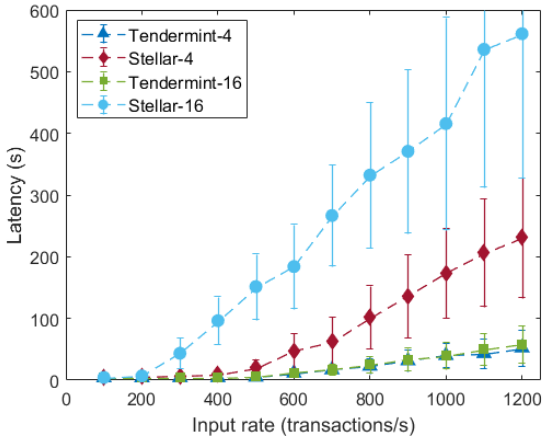
The latency of a transaction is the time between when the transaction is submitted by a client and when the transaction is committed in a block on a side chain, running either Tendermint or Stellar. Each transaction j may encounter different latency measures in V validators' databases, Latency_i^j, $\forall p_i \in V$. We thus denote $\text{Latency}_{\text{med}}^j$ as the median latency of transaction j among all V validators. The average latency of all the T transactions denoted as $\text{Latency}_{\text{avg}}$ can then be written as

$$\text{Latency}_{\text{avg}} = \frac{\Sigma_{j=1}^{T} \text{Latency}_{\text{med}}^{j}}{T}. \tag{2}$$

[4]Hashgraph does not provide open-source release. Hotstuff's open-source code is not free of bugs as of Q4 2019 to put into production.



(a) Throughput vs. Input Rate.



(b) Latency vs. Input Rate.

Fig. 2: Performance of Tendermint and Stellar with 4 and 16 Nodes (Validators).

Max block size	Trans. size	Commit interval
3,000 transactions/ block	212 bytes	5 seconds

TABLE II: Parameter Settings for Tendermint and Stellar.

B. Parameter Settings

Several parameters affect the block commit time, including maximum block size, transaction size, commit interval, transaction weight, and quorum-slice size [10], [36].

- *Maximum block size*. The maximum number of transactions that can be included in a block.
- *Transaction size*. The capacity of a transaction in bytes.
- *Minimal commit interval*. The minimum time increment between consecutive blocks. A minimum commit interval may prolong block commit time if an implementation intends to produce a block whose latency is smaller than the commit interval. However, we assume that a block's latency, which will be no longer than the commit interval, is an acceptable delay.
- *Transaction weight*. The probability of a transaction being committed in a block. All transaction types are given the same frequency in our experiments.
- *Quorum slice*. In Stellar, each validator has its quorum slice [28]. To have a fully connected network for our side-chain experiments, each validator's quorum slice is comprised of the rest of the validators so that all validators join the quorum.

We used the parameter settings listed in Table II to evaluate Tendermint and Stellar.

Both implementations were issued the same workload of a large number of synthetic transactions. To simulate realistic client behavior, we used a separate node to submit transactions to validators continuously. This node sent transactions in different input rates up to 1,200 transactions/s in a round-robin fashion.

C. Side Chain Evaluation

Figure 2 presents the performance of both implementations with 4 and 16 validators (or CPU nodes). (Later in the scalability study, we present performance with a larger number of nodes.) With 4 validators (or nodes), Tendermint and Stellar have nearly the same performance when the input rate is under 300 transactions/s (tps). However, when input rates exceed 400 tps, Tendermint achieves higher throughput and lower latency. The gap between the two protocols widens as the input rate is further increased.

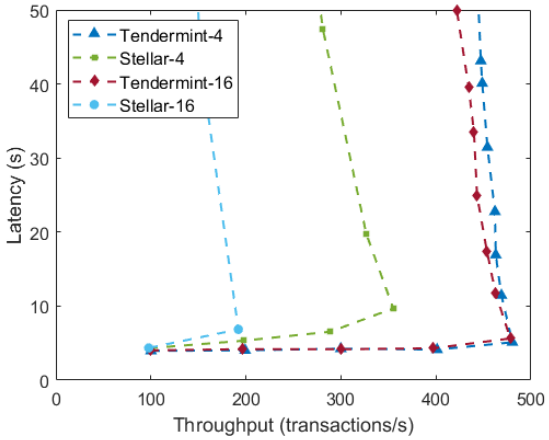
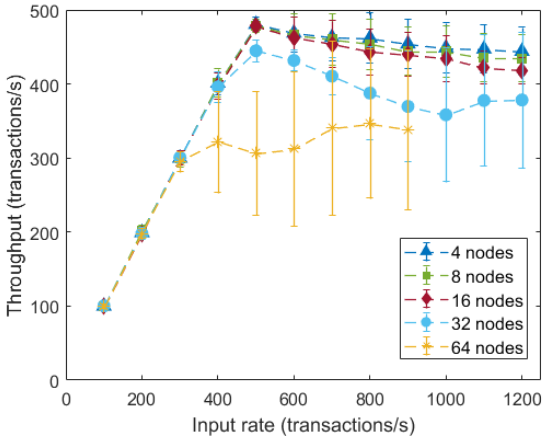
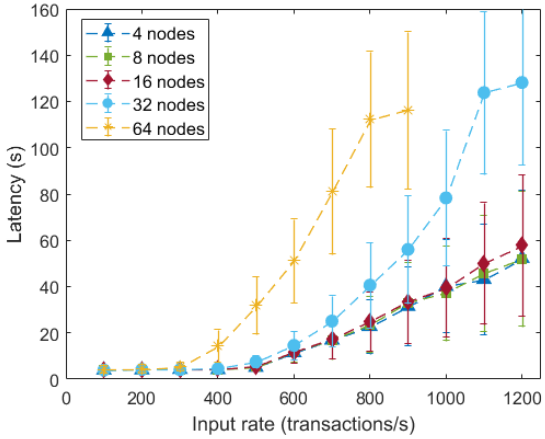


Fig. 3: Throughput vs. latency of Tendermint and Stellar with 4 and 16 nodes. In each dashed line, the consecutive vertices represent the input rate increases, starting from 100 tps and increasing by 100 tps each time.

Furthermore, we examine peak performance, which is defined as the highest throughput with the lowest latency. In Figure 3, we plot throughput (x-axis) and latency (y-axis) tradeoff for 4 and 16 validators of Tendermint and Stellar, respectively. The nodes on each line represent the input rates from 100 tps onward by increments of 100 tps. As an example,



(a) Throughput vs. Input Rate.



(b) Latency vs. Input Rate.

Fig. 4: Tendermint Performance with 4 to 64 Nodes.

for Stellar-4 (Stellar with 4 validators), its throughput initially steadily increases from an input rate of 100 to 400 tps, but then begins to degrade as the input rate increases beyond 400 tps. (Throughput starts to degrade when the curve(s) in the figure bends leftward as the input rate continues to increase.) Meanwhile, latency grows rapidly (beyond the upper bound of the figure after input rate is larger than 600 tps) when throughput starts to degrade due to saturation of system resources.

Having confirmed the latency and throughput tradeoff, we further stress Soteria with heavier workload and increase the number of validators to investigate problems and devising remedies. Figure 4 depicts Tendermint's through and latency at input rates increased from 200 to 1,200 tps. Stellar's performance exhibits the same patterns as Tendermint's, and we do not separately present its data. We report Tendermint vs. Stellar comparison in Figure 5.

From Figure 4[a], we observe that the throughput of Tendermint eventually saturates after the input rate has reached the capacity of the system. Throughput degradation occurs at a lower input rate when the number of validators are larger. The degradation point of 16-node Tendermint is at input rate 450 tps, while the degradation point of 64-node configuration starts at 250. Figure 4[b] depicts latency versus input rate. At the same input rate when throughput starts to degrade, the latency of Tendermint also increases drastically. For smaller configurations from 4 to 16 nodes latency reaches 50 seconds at input rate 1,200, whereas for larger configurations 32 and 64 latency reaches two minutes.

It is expected that system capacity eventually limits throughput and latency. To improve overall Soteria throughout at a guaranteed level of latency, we can configure an up to 32-node side chain with an admission control that limits the input rate to be under 450 tps. This configuration can support throughput up to 450 tps with 10-second latency. As we mentioned in Section III-C, Soteria can spawn multiple independent side-chain instances to improve throughput, while maintaining the same latency. When the overall throughput requirement is higher than 450, Soteria can configure another 32-node side chain to satisfy throughput scalability at the same latency.

Another avenue to simultaneously improve both latency and throughput is to use a small set of validators. This small jury approach is safe only if all validators are trusted and their systems highly secured and independently operated (in terms of system failure and security breach). One example scenario is in patient-data sharing. If a jury pool consists of the NIH, a small number of reputable hospitals and non-profit organizations, the jury pool can be trusted though small.

D. Two-Layer End-to-End Evaluation

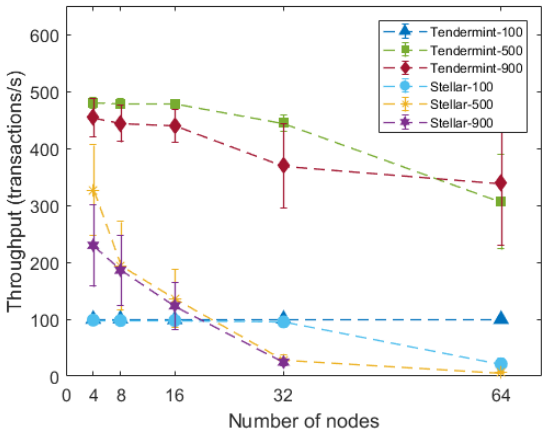
We have this far evaluated side-chain performance with two consensus protocols. We next examine Soteria's end-to-end performance with the main chain running the Ethereum protocol and one side chain running the Tendermint protocol.

Soteria collects on the side chain a number of blocks, each consists of a number of executed transactions, and then writes the fingerprint (or hash) of the blocks onto the main chain. The question is how many blocks on the side chain should be bundled to hash onto the main chain?

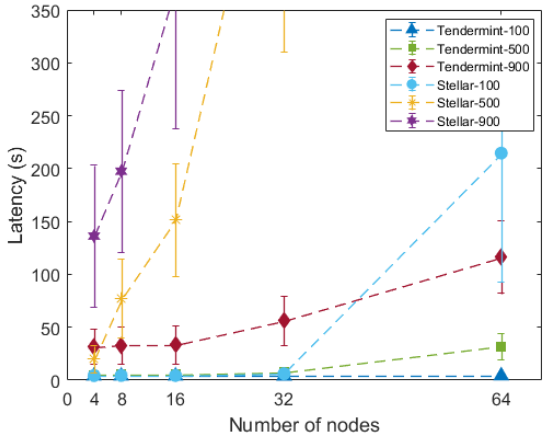
At first glance, higher frequency of inter-chain updates lowers the transaction latency. Our experiment, however, shows otherwise, as the bottleneck is on the main chain, not side chains. We will explain why the latency incurred on the main chain does not practically affect the design goals of Soteria after the experimental results are presented.

Figure 6 depicts that when the number of blocks per fingerprint is small (on the x-axis) or the inter-chain update rate is high, the end-to-end latency of Soteria can be as long as two minutes. The figure shows that this long latency is caused by the main chain, not by the side chains. The reason is that a public chain such as Ethereum is shared by many third parties, and the main chain's policy cannot be adjusted by Soteria. With high inter-chain update frequencies, Soteria can overload the main chain and cause lengthened delay.

When we reduce the inter-chain update rate to be once every 10 blocks or more, the latency is much reduced and stable (with a low variance). High update rates increase not only



(a) Throughput vs. # of Nodes.



(b) Latency vs. # of Nodes.

Fig. 5: Performance of Tendermint vs. Stellar at Input Rates 100, 500, and 900.

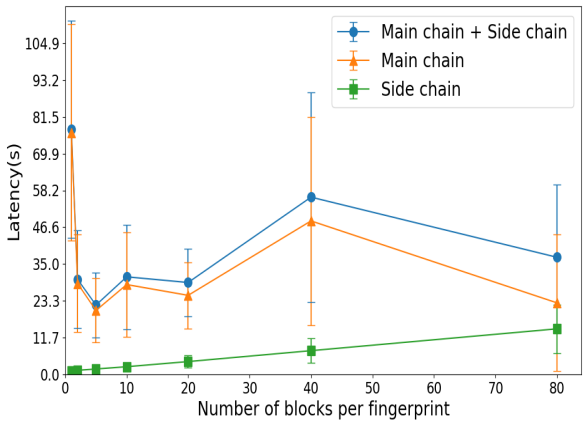


Fig. 6: Latency vs. Blocks per Fingerprint.

latency, but also cost, because the public main chain collects fees based on the number of updates. Soteria thus should set an update rate that can balance latency and cost.

E. Key Takeaways

From the results, we make the following observations:

- The Main chain must be decentralized for transparency to satisfy the provability of compliance with privacy regulations. Side chains can each independently run its own selected BFT/PBFT protocol and inter-operate with the main chain.
- The main chain's latency could be a performance bottleneck. Nevertheless, the purpose of the main chain is for public auditing, and a latency of minutes and even hours is acceptable by CCPA and GDPR user-privacy regulations. Moreover, the ATS module of Soteria can cache pending main-chain requests to perform first a fast audit and then later a confirmation once the main chain has hardened the side-chain's block hash.
- Side chains are permissioned and private, they can deploy BFT/PBFT protocols with a small jury pool to reduce latency. Transaction latency on side chains can thus be kept below ten seconds with an admission control. Throughput can be improved by spawning additional side-chain instances.
- Inter-chain update frequency can be dynamically adjusted depending on the workloads on both the main chain and side chains to balance between latency and cost.

V. Conclusion

This paper presented Soteria, consisting of a user right management system (URM), a distributed ledger (DLT), and a auditing service (ATS) to support provable, auditable and scalable data governance. To protect consumer rights of privacy and to comply with data-privacy regulations, we design Soteria to fulfill the functional requirements of consent, record keeping, and transparency (for auditing). We presented the architecture of Soteria, its functional specifications, and protocol choices for its base chain and side chains. On protocol selection, we qualitatively evaluated four algorithms and experimented with two readily deployable protocols Tendermint and Stellar. From studying the experimental results, we recommend side-chain and inter-chain configuration strategies to improve both latency and throughput.

Soteria cannot prevent a business from unlawfully distributing user data to a third party. However, when suspicion arises, Soteria can help validate data ownership, prove absence of user consent, and provide a list of potential unlawful distributors to facilitate an investigation. IP and data privacy protection involves substantial legal knowledge. Soteria shows that it is technically feasible to develop a scalable and public auditable distributed ledger. For additional challenges in judicial procedures to prove data rights please consult [23], [33].

Our future work will address various performance optimization issues that will arise with URM and DLT. Three specific topics are:

- SQL optimization. In Section II-C, we note that the SQL query, while correct, might not be very efficient, as it includes at least one clause for each owner of the data in the database. In our experience, this is sufficient, as in practice the same sharing agreement is used by many different consumers, so clauses can be unified when the query is constructed. As consumers' preferences become more fine-grained, this might not be the case. Future work should investigate an optimization algorithm or an index structure suitable to queries of this form, so that both performance and formal correctness can be maintained.
- Inter-chain protocol optimization. How often side chains should hash a block onto the main chain affects Soteria's latency and throughput. While Soteria may have full control on the parameters of its side chains, the main, public chain (such as Ethereum classic) can be shared with other applications. Soteria should look into dynamic adjustment policies on its side-chain block size and hash (to the main chain) frequency.
- Interoperability with native access control policies. Most companies have had their own access control policies and/or tools. For instance, Zelkova [6] developed by AWS is an SMT-based (Satisfiability Modulo Theories) reasoning tool for analyzing security/privacy policies and their risks. We will investigate how Soteria can complement existing tools.

VI. Acknowledgement

We would like to thank professor David Mazières (Department of Computer Science, Stanford University) for his insights on Stellar consensus protocol and valuable comments and suggestions to this work.

References

[1] General Data Protection Regulation (GDPR). Retrieved February 4, 2020, from https://gdpr-info.eu, 2016.

[2] Health Information Privacy Act, HIPAA. Retrieved February 4, 2020, from https://www.hhs.gov/hipaa/for-professionals/index.html, 2017.

[3] *NEM—Distributed Ledger Technology.*, 2018.

[4] AB-713 California Consumer Privacy Act (CCPA). Retrieved February 5, 2020, from https://leginfo.legislature.ca.gov, January 2020. California Legislature 2019-2020 Regular Session.

[5] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Fault-scalable byzantine fault-tolerant services. *SIGOPS Oper. Syst. Rev.*, 39(5):59–74, Oct. 2005.

[6] J. Backes, P. Bolignano, B. Cook, C. Dodge, A. Gacek, K. Luckow, N. Rungta, O. Tkachuk, and C. Varming. Semantic-based automated reasoning for aws access policies using smt. In *2018 Formal Methods in Computer Aided Design (FMCAD)*, pages 1–9, Oct 2018.

[7] L. Baird, M. Harmon, and P. Madsen. Hedera: A Governing Council & Public Hashgraph Network. Retrieved January 10, 2020, from https://www.hedera.com/hh-whitepaper-v2.0-17Sep19.pdf, 2018.

[8] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. *Handbook of satisfiability*, 185:825–885, 2009.

[9] E. Brewer. Towards Robust Distributed Systems. In *Proc. 19th Ann. ACM Symp.Principles of Distributed Computing (P)*, PODC 2000, pages 7–10, New York, NY, USA, 2000. ACM.

[10] E. Buchman. Tendermint: Byzantine Fault Tolerance in the Age of Blockchains. Master's thesis, 2016.

[11] E. Buchman, J. Kwon, and Z. Milosevic. The latest gossip on BFT consensus, 2018.

[12] V. Buterin and V. Griffith. Casper the Friendly Finality Gadget. *ArXiv*, ArXiv: 1710.09437, 2017.

[13] G. Campagna, J. Seo, M. Fischer, and M. S. Lam. Thingtalk : A distributed language for a social internet of things. In *Stanford OVAL Technical Report*, 2016.

[14] G. Campagna, S. Xu, M. Moradshahi, R. Socher, and M. S. Lam. Genie: A Generator of Natural Language Semantic Parsers for Virtual Assistant Commands. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 394–410, New York, NY, USA, 2019. Association for Computing Machinery.

[15] G. Campagna, S. Xu, R. Ramesh, M. Fischer, and M. S. Lam. Controlling fine-grain sharing in natural language with a virtual assistant. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(3):1–28, September 2018.

[16] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, page 173–186, USA, 1999.

[17] J. Chen and S. Micali. ALGORAND: the efficient and democratic ledger. *ArXiv*, ArXiv:1607.01341, 2016.

[18] J. A. Cowling, D. S. Myers, B. Liskov, R. Rodrigues, and L. Shrira. Hq replication: a hybrid quorum protocol for byzantine fault tolerance. In *OSDI '06*, 2006.

[19] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM*, 32(2):374–382, 1985.

[20] W.-K. Fu, Y.-S. Lin, G. Campagna, C. Liu, D.-Y. Tsai, C.-H. Mei, E. Y. Chang, S.-W. Liao, and M. S. Lam. Soteria: A provably compliant user right manager using a novel two-layer blockchain technology (extended). *Stanford Technical Report http://infolab.stanford.edu/~echang/SoteriaExtended.pdf*, August 2020.

[21] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu. SBFT: a Scalable and Decentralized Trust Infrastructure, 2018.

[22] S. King and S. Nadal. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. 2012.

[23] J. Kishigami, S. Fujimura, H. Watanabe, A. Nakadaira, and A. Akutsu. The blockchain-based digital content distribution system. In *IEEE Int. Conf. on Big Data and Cloud Computing*, pages 187–190, 2015.

[24] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative byzantine fault tolerance. *SIGOPS Oper. Syst. Rev.*, 41(6):45–58, Oct. 2007.

[25] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[26] S. Liao, E. Y. Chang, C. Liu, W. Lin, P. Liao, W. Fu, C. Mei, and E. J. Chang. DeepLinQ: Distributed Multi-Layer Ledgers for Privacy-Preserving Data Sharing. In *2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, pages 173–178, 2018.

[27] M. Lokhava, G. Losa, D. Mazières, G. Hoare, N. Barry, E. Gafni, J. Jove, R. Malinowsky, and J. McCaleb. Fast and Secure Global Payments with Stellar. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, pages 80–96, 2019.

[28] D. Mazières. The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus, 2015.

[29] J. Mickens. The saddest moment. *login Usenix Mag.*, 39, 2014.

[30] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The Honey Badger of BFT Protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 31–42, New York, NY, USA, 2016. Association for Computing Machinery.

[31] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. *Cryptography Mailing list*, 2009.

[32] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, June 2014. USENIX Association.

[33] A. Savelyev. Copyright in the blockchain era: Promises and challenges. In *Computer law & security review*, volume 34, pages 550–561, 2018.

[34] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung. Spin one's wheels? byzantine fault tolerance with a spinning primary. In *2009 28th IEEE International Symposium on Reliable Distributed Systems*, pages 135–144, 2009.

[35] M. Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In J. Camenisch and D. Kesdoğan, editors, *Open Problems in Network Security*, pages 112–125, Cham, 2016. Springer International Publishing.

[36] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. HotStuff: BFT Consensus in the Lens of Blockchain, 2018.