Study on Integrity and Privacy Requirements of Distributed Ledger Technologies

Sachiko Yoshihama  
IBM Research – Tokyo  
Tokyo, Japan  
sachikoy@jp.ibm.com

Shin Saito  
IBM Research – Tokyo  
Tokyo, Japan  
shinsa@jp.ibm.com

Abstract— Distributed Ledger Technology (DLT) allows multiple organizations to share transaction data while protecting data integrity without the need for a central authority. However, there is a strong need to protect the privacy of transactions from parties that are not stakeholders of each transaction. Since DLT is a technology for sharing replicas of the same ledger data across multiple parties, protecting confidentiality and data privacy is essentially difficult. In this paper, we first define the properties of integrity and privacy requirements and discuss the requirements in typical use cases. Finally, we review how the requirements are addressed in existing implementations.

Keywords—Blockchain, Distributed Ledger Technology, Transaction Systems, Security, Privacy

I. INTRODUCTION

In 2008, an author known as Satoshi Nakamoto published a paper [5] on a cryptographic protocol that realizes a virtual currency without the need for a central authority. The proposal has been implemented and operated as a cryptocurrency called Bitcoin, which has gained wider attention in industries and academia. The baseline technology that enables Bitcoin is called blockchain, or distributed ledger technology (DLT), and it has been applied to a wider range of business use cases not only in financial industries but also in others.

Privacy protection is one of the major concerns when applying DLT in business use cases. The key concept of DLT is that multiple participants share a replica of the same ledger, and they keep adding records to the ledger by reaching a consensus among each other. In DLT, participants share a logically identical ledger in its replicated form so that any falsification of one of the replicas may be easily detected by comparing the replicas. Therefore, DLT is often called a tamper resistant technology because the technology is designed to protect the integrity of ledgers when not all participants can be trusted. To successfully tamper with a ledger, a large enough number of participants have to collude, depending on the consensus algorithm. However, it is essentially difficult to protect the confidentiality and privacy of a ledger due to the nature of the shared replications.

In this paper, we first define the functional components of DLT. Then, we define the integrity and privacy properties and assess examples of requirements in real-life use cases. Finally, we make observations on several typical DLT implementations and discuss how they satisfy the requirements.
A smart contract is often used to perform state transition by updating states on the basis of logic. However, some implementations (such as Corda [11]) use smart contracts in combination with UTXO to add rules in the case of making a decision when transferring asset ownership.

C. Consensus

In DLT, to maintain consistency and soundness among the nodes that participate in a network, the nodes have to come to an agreement. In this paper, we use the term consensus to refer to such activities. Consensus methods can be classified into two types: proof-of-work and its variations and distributed consensus algorithms.

1) Proof-of-Work and its Variations

Simply put, proof-of-work (PoW) is a kind of computation competition among nodes, the goal of which is to find one of the correct values that meets a required condition. The node that finds a correct value has the right to add a new block, approve a set of transactions in the block, and then receive a virtual currency as a reward. With Bitcoin, the nodes try to find a nonce, where the cryptographic hash value of the nonce, hash of transactions in a block, and the hash of the previous block become smaller than a pre-agreed on target value. A cryptographic hash is a one-way function that has no effective way to estimate the input from a given output. Therefore, each node has to perform brute-force try-and-error computations to find a correct nonce in a process known as mining. Thus, a node or a set of collaborating nodes with larger computing resources can have a higher probability of winning a mining competition.

There are variations of PoW, such as proof-of-stake (PoS) or proof-of-importance (PoI), that change the probability of a win with not only computation power but with other properties, such as the amount of the virtual currency of each node or importance of a node as judged from the amount of transactions it participates in.

Well-known shortcomings of PoW include insufficient safety and a lack of finality. In this context, safety refers to resistance to attacks made by adversaries in a blockchain network, such as an attempt to approve illegal transactions. Such attackers may win a mining competition at some point, add an illegal, e.g., double-spending, transaction to a block, and disseminate the block to other nodes in the network. The Bitcoin paper [5] claims that safety will increase over time as many blocks are added to the chain because the probability that an attacker will continuously win in the mining is small, and other correct nodes will reject blocks that contain illegal transactions. However, if multiple participants collude and own more than 50% of the computation power in the network, they can approve arbitrary transactions.

Finality is a concept used in the financial industry. It means that once a settlement or money transfer is completed, it is “irrevocable and unconditional”[4]. PoW and its variations cannot guarantee finality because even if a node succeeds in mining and then adds a block to the ledger, another node may succeed in mining and add a different block before the first block is well propagated in the network. In such a case, two or more versions of the ledger may co-exist in the network, creating a fork in the chain of blocks. If a fork occurs, a longer branch will eventually be deemed correct, and a shorter branch will be rejected because a branch that is supported by more mining nodes will have a better chance of adding new blocks and growing faster. Therefore, transactions that were recorded only in the shorter branch will be cancelled, and thus, finality is not guaranteed. Some attacks, such as the eclipse attack [12], attempt to isolate some nodes by polarizing the P2P network so that those nodes cannot receive the latest block, resulting in a fork. PoW and its variations are good mechanisms for reaching a consensus in a public blockchain network with anonymous nodes, and they are tolerant against some attackers or Byzantine fault nodes. However, the lack of safety and finality is deemed a critical issue when using DLT in real-world businesses.

2) Distributed Consensus Algorithm

While DLT is being tested and considered in business use cases, there has been increasing attention given to permissioned DLT technologies, where participants are authenticated and authorized. In a permissioned DLT, the number of nodes and their identities can be controlled, so it is possible to use traditional distributed consensus algorithms that are based on message exchange. Paxos [13], Viewstamped Replication (VSR)[14], Raft [15], and Practical Byzantine Fault Tolerance (PBFT)[6] are well known classic algorithms that were developed for state machine replications. These distributed consensus algorithms assume asynchronous message communication called the eventual-synchrony network model. Typically, after some messages are broadcasted in a network, a consensus is reached, and messages are then delivered to the correct nodes. When a consensus mechanism satisfies the following properties, it is called atomic broadcast [9].

- Validity: If a correct node p broadcasts a message m, p eventually delivers m.
- Agreement: If a message m is delivered by some correct node, m is eventually delivered by every correct node.
- Integrity: No correct node delivers the same message more than once.
- Total Order: Suppose p and q are two correct nodes that deliver messages m1 and m2 respectively; then, p delivers m1 before m2 if and only if q delivers m1 before m2.

III. INTEGRITY AND PRIVACY REQUIREMENTS OF DLT

One of the characteristics that differentiates DLT from classic state replication is its assumption on the environment. DLT assumes that network participants span across multiple entities and organizations that have a conflict of interest and potentially include hostile entities. An advantage of DLT is that it can guarantee the integrity of data in such an environment because each of multiple participants owns an identical copy of the ledger, and thus, the tampering of adversaries can be detected by comparing replicas of correct nodes. When a Byzantine fault tolerant consensus mechanism is used even when some of the participants behave maliciously the network can keep functioning correctly.

A Byzantine fault tolerant consensus algorithm typically defines a safety property such that, in a network where
independent $N$ nodes participate in reaching a consensus, a consensus can be reached as long as $f$ nodes or fewer are in Byzantine failure.

However, since the ledger is shared among nodes, protecting the privacy of the ledger is essentially difficult. This is problematic in some use cases, e.g., in equity trading, because it is a common requirement that the details of a trade be shared only between the counterparties of the trade, i.e., seller and buyer, and not visible to other third parties.

To overcome the privacy issue, some algorithms reach consensus not for all the nodes in a network but only between $N_{cp}$ counterparties, e.g., $N_{cp} = 2$ when the counterparties are only a seller and a buyer, and thus, the data has to be shared only between counterparties. However, if one of the two parties becomes Byzantine faulty when the data is shared only between two parties, protecting integrity is difficult. Intuitively, as the number of parties involved in the consensus increases, the integrity protection will strengthen and vice versa.

In this section, we define several properties of integrity as well as confidentiality and privacy requirements.

A. Integrity

Let $n_1, n_2, ..., n_b$ be the independent nodes in a DLT network, and each node $n_i$ keeps a copy of the ledger $L_i$. The ledger $L_i$ owned by a node $n_i$ is expressed as follows.

$$L_i = \{ h(tx_{i1}), h(tx_{i2}), ..., h(tx_{im}) \}$$

Here, $tx_{ij}$ is the $j^{th}$ transaction in $n_i$'s ledger, and $h(tx_{ij})$ is the cryptographic hash value of it. Now, in addition to the properties of the atomic broadcast described in the previous section, there are integrity requirements as follows.

- **11. Agreement on Transaction Validity:** This is a property for which only a legitimate transaction can be recorded in the ledger, depending on the transaction semantics. For instance, a double-spending transaction or remittance that result in a negative account balance are examples of invalid transactions that should be rejected. A decision on the validity of such transactions should be consistent among $N-f$ or more nodes, where $f$ is the utmost number of faulty nodes. In other words, for independent correct nodes $n_i$ and $n_j$, let $valid(tx_{ij})$ and $valid(tx_{ji})$ be the decision made by node $n_i$ and $n_j$, respectively, whether transaction $tx_{ij}$ is valid or not. Then, transaction validity is guaranteed iff $\forall i \in P_{correct}$ $\bigwedge \forall tx \in L_i valid(tx)$, where $P_{correct}$ refers to all correct nodes.

- **12. Tamper Evidence:** It is often required that the ledger be not tampered with and consistent among participants. If replicas of the ledger owned by some of the nodes are illegally modified, that should be detectable. To detect the tampering of a transaction, one needs to check and compare only the hash value of transaction $h(tx)$ and does not need to see $tx$ itself. We define DLT as being tamper evident if a node can compare the transaction hash value with other nodes to detect tampering.

- **13. Finality:** As described in the previous section, finality refers to a property for which no transaction will be rejected after it is approved and recorded in the ledger. In other words, for any of the correct nodes $n_i$, the DLT guarantees finality iff for all $tx_v$ in $L_i$ at time $t$, so this is the same as $tx_v$ in $L_i$ at $t'$ if $t < t'$.

Note that when finality is not guaranteed, the content of a ledger may change over time, i.e., an agreement on transaction validity that is satisfied at some point in time may be overthrown later.

B. Confidentiality and Privacy

For the sake of simplicity, in this section, we model the privacy properties in bilateral transactions. Let $n_1, n_2, ..., n_b$ be the nodes in a DLT network, and let there be an arbitrary number of subjects $S = \{ s_1, s_2, ... \}$, e.g., a subject can be a human user in a business entity, that trade with each other. Subjects access DLT via nodes. When two subjects $s_p, s_q$ transact via nodes $n_p, n_q$, respectively, and $m$ is the message, each transaction $tx$ can be represented as $(s_p, s_q, n_p, n_q, m)$. We also assume that subject $s_p, s_q$ can be anonymized, and the real identity of each subject is represented as $ID_p, ID_q$ respectively. There are possible privacy properties as follows.

- **C1. Anonymity to Third Parties:** The identities of the trading subjects are not visible to parties other than the counterparties of the trade, i.e., each of $n_p, n_q, s_p, s_q$ can see the true identity of subjects such as $s_p = ID_p$ and $s_q = ID_q$. However, no third party $s_r \neq s_p, s_q$ can see the true identity of subjects.

- **C2. Anonymity between Counterparties:** In addition to C1, the identities of trading subjects are not visible even between the counterparties. That is, neither $n_p$ nor $s_q$ see $s_q = ID_q$, and neither $n_p$ nor $s_r$ see $s_r = ID_r$. A typical example that has this property is remittance to an anonymous identity in a virtual currency.

- **C3. Confidentiality of Transaction Content:** We refer to the details of a trade as the transaction content, such as the name of the asset or amount to be traded. The confidentiality property of the content can be defined as follows. For any transaction $tx = (s_p, s_q, n_p, n_q, m)$, third party subject $s_p, s_q$, or third party nodes $n_k \neq n_p, n_q$ can see the content of $tx$. However, $s_p$ and $s_q$ may see the transaction hash value $h(tx)$ because the value does not disclose the transaction details.

- **C4. Confidentiality about Existence of Transaction:** This is a property that ensures that the fact that a transaction $tx$ occurred is not visible to a third party subject $s_p, s_q$, or a third party node $n_k$ where $n_k \neq n_p, n_q$.

- **C5. Confidentiality of State:** As described earlier, state refers to the latest state of assets that is the result of transaction invocations. The confidentiality of a state can be defined as follows. When a subject $s_k$ has a state variable $v$ that it would like to share only with subject $s_l$ and nodes $n_k, n_l$, no subject $s_j, s_p, s_q$ or third party nodes $n_k \neq n_l, n_k$ should be able to see $v$.

IV. INTEGRITY AND PRIVACY REQUIREMENTS IN TYPICAL DLT USE CASES

In this section, we review several real-world DLT use cases and discuss their integrity and privacy protection mechanisms.

A. Virtual Currency

Virtual currencies are the most typical use cases of DLT. Starting from Bitcoin, there are hundreds of virtual currency implementations on DLT. As an attempt to reproduce cash in the digital world, use cases are focused on the transfer of simple values, i.e., coins, in a public network. For historical reasons, most implementations use UTXO and reach consensus with PoW or its variations.

The integrity requirements of virtual currencies typically include transaction validity (11) and tamper resistance (12). As the use case is essentially the same as a financial settlement, finality (13) should also be a latent requirement. However, because most virtual currencies are PoW-based implementations, finality cannot be guaranteed. Risk of no-
finality is mitigated by operational customs, such as waiting for long enough after a transaction is recorded in a block and several new blocks are added after it.

Privacy requirements for anonymity (C1 and C2) are satisfied because each subject is represented by an anonymous address derived from an arbitrary cryptographic public key. However, since all nodes perform a transaction validity check in PoW, transaction content (C3) or its existence (C4) are disclosed to all parties. Most virtual currencies do not have states, so the confidentiality of a state (C5) is not applicable. However, if an implementation has states, protecting its confidentiality is difficult as long as the DLT is operated as a public network.

B. Asset Management

Some DLT use cases address asset management. For example, ownership of tangible assets, e.g., vehicles and real-estate, or intangible assets, e.g., digital content, is recorded in the ledger and traded on it [16]. In a typical implementation, an asset is represented as a token, and ownership of the token is transferred by using UTXO. In asset management, integrity properties such as transaction validity (I1), tamper resistance (I2), and finality (I3) are required.

Privacy requirements will depend on the type of asset being managed. For example, in Japan, real estate registration is made public, and anybody can obtain a copy if he or she requests [17]. In such a case, all privacy properties C1–C5 will not be required, except the privacy of an individual asset owner.

C. Post-Trade Processes in Equity Trading

In financial markets, post-trade processes refer to the processes of clearing and settlement after sell and buy orders are matched and a trade is executed. Several proof-of-concept trials have been conducted to assess the technical feasibility of transforming these processes by using DLT [7]. Post-trade processes need to handle complicated state transitions of equities and funds that span across multiple asynchronous external events.

One of the strong requirements of the post-trade process is delivery-versus-payment (DVP), which is a common practice in the settlement of an equity trade. DVP ensures that, for each trade, the settlements of both equity and funds have to be completed. If the settlement of a fund fails, that of equity should fail too and vice versa. In addition, the post-trade process also has to perform complicated calculations, such as the netting of multiple settlements between two parties, i.e., bi-literal netting, or across multiple parties, i.e., multi-lateral netting.

Therefore, to properly implement post-trade processes on DLT, smart contracts and states are necessary functions because UTXO is too simple for implementing the aforementioned complicated processes. Since this use case typically handle a large volume of funds or equities, transaction validity (I1) and tamper resistance (I2) are quite important. In addition, as is the nature of financial systems, there is a very strong requirement for finality guarantee (I3).

Since financial markets are a highly regulated industry, we can assume that (at least in the foreseeable future) DLT will be used in a permissioned network among authorized financial institutions, such as banks and clearing organizations. In such a case, it should be acceptable that only a small number of nodes (such as only Ncp counterparties) participate in consensus, minimizing the exposure of the transaction content or state to third parties that are not involved in each transaction.

To check the validity of a transaction, some information flow has to occur with third parties other than trading counterparties. For example, if a subject sp is transacting via n1 and bilaterally trading with each of multiple parties such as sq and sj that are transacting via n1 and nj, respectively. Then, the account balance of sp has to be checked across counterparties. This is because the account balance of sp may change as it trades in each of the bilateral relations; e.g., let us say that sp has 10 shares of equity in its account and sells 8 shares to sq and then 3 to sj. In the first trade, nodes n1 and nj check the transaction validity to ensure that sp has enough shares in the account, and the transaction then transfers ownership of 8 shares to sq. Then, after the trade between sp and sq is completed, the new account balance (10–8=2) has to be informed to nj so that both n1 and nj can check the transaction validity for the second trade that attempts to transfer 3 shares from sq to sj. Then, there is the information flow from the first trade to the second trade, which allows nj to infer some part of the transaction content of the first trade, to which nj is a third party.

Therefore, even if the trust assumption does not require all parties to own all of the accounts’ states, the system requires the accounts’ data to be shared across parties, beyond each transaction’s counterparts.

In financial trading, regulatory requirements on privacy protection are quite strong. In particular, when two financial institutions are trading with each other, the content of the trade and the fact of the trade occurring need to be concealed. For example, in equity trading, the price of equity may change significantly if the trade information is leaked in advance. One may also take advantage of such information to make a profit by estimating the equity price. Therefore, anonymity to the third party (C1), transaction content confidentiality (C3), and state confidentiality (C5) are strong requirements in this use case. However, satisfying C5 is difficult if an account balance has to be shared across parties.

Since the counterparts are often known and identified financial institutions, the anonymity of counterparts (C2) can be excluded from the requirements. However, an individual investor’s information should be protected if it is ever recorded in a ledger. It is also desirable to conceal the transaction existence (C4), but it should be acceptable as long as K-anonymity is held for a large enough K.

D. Derivative Contract

A derivative contract or securitized financial products are a kind of financial contract that can be implemented on a smart contract. Let us take the Equity Linked Note (ELN) as an example, which is a securitized financial product of which terms and conditions are linked to the price of a pre-defined equity and a financial derivative called an option. ELN is issued at a discount price to the principal amount of the corresponding equity. On a pre-agreed on valuation date, if the equity’s price is above or equal to the strike price, the investor will then
receive the principal amount at the maturity date. Otherwise, the investor is exposed to the equity’s negative performance. Such derivatives do not trade the actual equities, but rather, a set of contract terms and conditions are pre-agreed on, and the settlement amount and conditions change on the basis of external events such as equity price changes. Such a use case requires that trade logic be encoded and executed as a smart contract while maintaining the states.

In a financial contract, transaction validity (I1) and tamper resistance (I2) are very important, and as a financial system, there is a strong requirement for finality (I3).

As a derivative contract is essentially a contract between the buyer and seller, it is acceptable that consensus is reached only between the trading counterparties. In addition, unlike the settlement of equities, there is no need to manage account balances, and thus, it is ok that the data is shared only between trading counterparties. However, other parties may need to access the data in the case of an audit or dispute resolution. As for the privacy requirements, the same set of properties as equity trading, i.e., C1, C3, C4, and C5, will be required.

V. REPRESENTATIVE DLT IMPLEMENTATIONS
In this section, we review several popular DLT implementations and observe how each of them can satisfy the integrity and privacy properties. A summary is shown in:

A. Bitcoin
With Bitcoin [5], the only data in the ledger is blocks, and transactions are processed with UTXO. Consensus is reached by using PoW, and thus, the integrity of an instance of the ledger cannot be guaranteed at some point in time. Due to the nature of PoW, Byzantine fault tolerance can be expressed not by the number of faulty nodes but in the amount of computation power in the DLT network. As long as the total computation power of faulty nodes \( f \) is \(< 2N\), the probability of the integrity of the ledger being assured will increase as time elapses, but finality cannot be guaranteed.

As for privacy requirements, since each subject transacts by using an address derived from an anonymous public key, privacy requirements C1 and C2 will be satisfied. However, since the content of the ledger will be visible for all nodes, Bitcoin cannot guarantee other privacy requirements, such as C3, C4, and C5.

B. Ethereum
Ethereum [3] manages ledger data as blocks and states. Transactions are processed in smart contracts, and transaction validity is checked by using application logic implemented in a Turing complete language called Solidity. Consensus is reached by using PoW; therefore, the same as Bitcoin, integrity requirements are satisfied only probabilistically, and finality is not guaranteed. As long as anonymous addresses are used for subject identities, privacy requirements C1 and C2 will be satisfied, but other privacy requirements will not be satisfied the same as Bitcoin.

In late 2017, a new privacy preserved currency called Zcash [19] was announced. It is an open source implementation of Zerocash [20] built on a type of zero-knowledge proof technology known as Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK)[21]. Although zk-SNARK has the potential to protect the privacy of a wide variety of smart contracts, the current system is not practical because of drawbacks, such as complexity in key management and expensive transaction fees, i.e., gas, for setting up a secret for each smart contract.

C. Quorum
Quorum [10] is a permissioned DLT based on Ethereum but modified to address the needs of business use cases. For instance, Quorum uses a voting-based consensus mechanism to replace PoW and added privacy protection capabilities. In Quorum, in addition to ordinary nodes called quorum nodes, each participant will have a transaction manager node that forms a constellation, a peer-to-peer encrypted message exchange for private transactions. Public transactions are processed similarly to public Ethereum and disseminated to all nodes. However, private transactions are encrypted and sent from a quorum node to the transaction manager and shared only between the transaction managers that are entitled to access the content of a private transaction. Quorum nodes only share the hash values of private transactions and not their content. When processing a private transaction, only entitled participants decrypt the transactions and execute a smart contract on Ethereum Virtual Machine (EVM) to check the validity of the transactions.

Quorum supports two consensus algorithms, QuorumChain and Raft [15]. For the sake of simplicity, this paper is focused on the former. In QuorumChain, each node may have up to two roles: a voter and a maker. A maker node can create a new block, while a voter can cast a vote on it. A node with no roles is deemed a passive observer.

A new block is created in the following steps.

1) A maker creates a new block, signs it, and disseminates it to other nodes by using the P2P protocol. Each block contains a list of transactions and votes on the previous block.

2) Upon the receipt of a block, each voter verifies whether the block was sent from a legitimate maker and executes the smart contract to process the transaction. Then, the voter calculates the hash value of the state that is the result of transaction execution and verifies that it is consistent with the hash value of its own state.

3) After verification is completed, the voter casts a vote by invoking a vote transaction in a special smart contract called BlockVoting. Since the vote itself is processed as a transaction, the result of consensus will be determined at the time that the next block is created.

4) If there are more votes than a necessary threshold before timeout, the maker decides that the block is successfully created. Otherwise, the maker retries voting by repeating the process from steps 1 to 3.

There has to be at least one maker in a network. If there are multiple makers, each maker waits for a random duration so that each maker a new block at a different timing without causing conflict.
Each voter casts a vote as long as it succeeds with block verification. Due to the possibility of a retry, each voter may cast more than one vote for the blocks with the same block number. Then, when a block has more votes than the required threshold, each node accepts the block and adds it to its own ledger. If there are multiple blocks with the same block number, the block with the most votes will be chosen.

In QuorumChain, if there is only a single maker, it could be a single point of failure. If there are multiple makers, the chain might fork if some makers are in Byzantine failure. If there is a relatively large communication delay between makers, multiple makers may create blocks with the same block number, and thus, finality cannot be guaranteed.

In the case of private transactions, transaction validity is checked by a set of nodes that are authorized to access the content of private transactions. However, nodes that are not authorized to access a private transaction can still participate in integrity verification because they have transaction hash values. Therefore, the integrity is tolerant to up to \( f \) nodes, where \( f < 2/N \).

Since private transactions are encrypted, privacy requirements \( C1 \) and \( C3 \) can be satisfied. However, trading counterparties can identify each other, and thus, \( C2 \) is not satisfied. In addition, any third party nodes will receive the hash of each transaction, and thus, \( C4 \) is not satisfied either. Since only counterparties can decrypt a transaction and store the resulting state, requirement \( C5 \) is satisfied.

### D. R3 Corda

Corda [11] does not have blocks, and states are atomic units of information. Each state represents an asset being traded and can be either spent or unspent. In addition, each state may have additional attachment files.

Transactions are processed in an extension of UTXO. Each transaction specifies input states, output states, and notaries. States can also be associated with logic defined in Java bytecode, which will be executed as a smart contract that checks the validity of a transaction.

Notary is a special node that verifies whether all input states are unspent, and thus, there is no illegal double spending. Each transaction can consume only the input states that belong to the same notary, and thus, a single specified notary can verify all input states. There is also a special transaction that spans across multiple notaries can be processed. A complicated transaction, which is performed in multiple steps by multiple parties, is encoded in a state machine called flow, for which a state transition is handled asynchronously through multiple events.

Each transaction is not broadcasted to all nodes but sent to the nodes that need to refer to the transaction in order to validate transactions. In addition, since there are no blocks, the total order of transactions is not defined. Only the partial order of dependent transactions is defined, where each partial order is guaranteed by a notary. Notaries do not depend on a specific consensus protocol. A notary can be a single node that performs the role of a trusted centralized service or multiple nodes that reach consensus by using protocols such as Raft or BFT-SMaRt.

With Corda, the parties involved in consensus are notary and trading counterparties, i.e., \( N_p \) nodes, and the DLT is not tolerant to Byzantine failure of these nodes. In particular, if the notary is a single node, it can become a single point of failure and create a fork if it is in Byzantine failure.

Privacy is addressed by avoiding global broadcasts, but there are still some uncertainties, i.e., when validating a transaction, each node will access dependent transactions. Therefore, each transaction should be seen only by nodes that are related. However, if a transaction has complicated dependencies on other transactions, reasoning about which nodes may see the transactions is difficult, even though it is still possible to protect anonymity by using anonymous signature keys [11].

### E. Hyperledger Fabric V0.6

Hyperledger Fabric [2] (Fabric, in short) is an open source DLT platform that focuses on permissioned networks. Fabric V0.6 is an experimental version released in 2016. Each node in Fabric V0.6 has both blocks and states. Transactions are processed by using a smart contract, which is defined in Turing complete programming languages such as Golang. The consensus mechanism is designed to be pluggable, but PBFT [6] (Figure 1) is the supported implementation. Because of the safety property of PBFT, it is tolerant to up to \( f \) nodes in Byzantine failure, where \( f \leq (N-1)/3 \). Under the same condition, integrity requirements \( I1 \) and \( I2 \) are satisfied, and PBFT guarantees \( I3 \). However, the implementation reaches consensus only before executing a smart contract, and no consensus is reached on the result of smart contract execution. Therefore, Fabric V0.6 depends on the assumption that each smart contract is completely deterministic.

![Hyperledger Fabric V0.6 Consensus Flow](image)

All participants (including users and nodes) are authenticated by using digital certificates called ECerts (enrollment certificates). However, the anonymity of subjects in a transaction is protected by using anonymous, one-time certificates called TCerts (transaction certificates). Therefore, anonymity requirements (\( C1 \) and \( C2 \)) can be protected, unless user identities are included in the transaction content. Optionally, content and state DB can be encrypted to protect privacy, but a malicious system administrator may steal the encryption key to decrypt the data. Therefore, to protect the privacy of the transaction content (\( C3 \)) and states (\( C5 \)), all nodes have to be under the control of a trusted administrator or protected in a trusted execution environment. For example, by
using hardware-level security mechanisms, we can use secure boot to run a node in a specific controlled configuration in a secure container, which prevents OS users from logging-in to the system to run arbitrary OS-level commands. Likewise, cryptographic keys can be protected by using hardware security modules. Privacy regarding the existence of transactions (C4) cannot be satisfied as the ledgers are shared across all participants.

F. Hyperledger Fabric V1.0

Hyperledger Fabric V1.0 is a new version that underwent significant architecture changes to overcome shortcomings identified in Fabric V0.6. Similar to Fabric V0.6, both blocks and states are shared across nodes. In Fabric V1.0, nodes have multiple roles; a node that manages ledgers is called a peer, and a node that orders transactions is called an orderer. A peer that executes a smart contract is called an endorsing peer or endorser.

Consensus is reached in the following process (Figure 2). First, a client application sends a transaction proposal to one or more endorsing peers. Each endorsing peer executes a smart contract to process the transaction and derives the execution result as a read-write set, i.e., the keys and data in the states being read and written as the result of smart contract execution. The read-write set and corresponding transaction content is signed by the peers to represent its endorsement and sent back to the client. The process is repeated (of course, it can run in parallel) until the client receives a sufficient enough number of endorsements as defined in the endorsement policy. Then, the client submits the transaction to an orderer, along with the endorsements from the peers. The orderer determines the total order of transactions and delivers a block of ordered transactions to the peers. Finally, each peer verifies the transactions, records them into the blocks, and updates the state according to the read-write sets. Each peer’s verification includes the validation of the endorsements’ signatures and whether each transaction has enough endorsements as defined in the endorsement policy. In addition, peers conduct multi-version concurrency control (MVCC) checks to make sure that no transactions override the write set of other transactions.

Each peer belongs to an organization and proves it by having a certificate that is issued by the root CA of the organization or by any intermediary CAs that can be traced back to the root CA. An atomic component in an endorsement policy is an organization, and a policy is defined by using an \( m \)-out-of-\( n \) rule. For example, a policy may state that a transaction requires at least two endorsements from three organizations \{A, B, C\}. The policy can be cascaded to give weights to each organization, e.g., a policy may state that a transaction always needs an endorsement from organization A and one of \{B, C\}.

Figure 2 Hyperledger Fabric V1.0 Consensus Flow

Depending on the endorsement policy, Fabric V1.0 is tolerant to the Byzantine fault of up to \( f < N/2 \) endorsing peers, where \( f < N/2 \). However, since a policy is defined on the basis of an organization unit, it has to be assumed that all peers in each organization have the same level of trust. In other words, when \( N \) is the number of organizations in a network, DLT is tolerant to up to the failure of \( f \) organizations, where \( f < N/2 \), where a failure of an organization is defined as the failure of any of the peers that belong to the organization. If \( N \) is big, there is a negative impact on performance from requiring endorsements from peers from many organizations because of the overhead of signature verification by peers. Therefore, in reality, a policy has to be defined by assuming a smaller number of failed organizations.

In Fabric V1.0, the orderers are implemented by using Apache Kafka, which can be clustered into multiple nodes to enable crash tolerance but not Byzantine fault tolerance. For a future version of Fabric, there are plans to implement a Byzantine-fault-tolerant ordering service that uses BFT-based algorithms.

Fabric V1.0 has a multichannel mechanism that makes it possible to create a private channel for a subset of peers in a network. The entirety of an information flow is confined within the channel, including shared data (blocks and state) and consensus reaching. Therefore, integrity and privacy requirements can be satisfied differently, depending on the use of channels.

First, let us focus on peers that belong to a single channel. Assuming that an endorsement policy is satisfied and an orderer is not in Byzantine failure, integrity requirements I1, I2, and I3 can be satisfied. However, the system is not tolerant to the Byzantine failure of orderers. As for the privacy requirements, since TCert is not supported in Fabric V1.0, transaction anonymity (C1 and C2) is not supported. Other privacy requirements C3, C4, and C5 can be satisfied under the same condition as Fabric V0.6.

If we set up a channel for each set of trading counterparties, transaction data and ledger data are isolated per each channel, and thus, privacy requirements C1–C5 are satisfied. Integrity requirements are satisfied the same as in the single-channel case, but care has to be taken when defining the endorsement policy so that the required \( f < N/2 \) or less holds per each channel.
Table 1 Summary of DLT Implementations

<table>
<thead>
<tr>
<th>Property</th>
<th>Ethereum</th>
<th>Corda</th>
<th>Hyperledger Fabric</th>
<th>Zcash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Structure</td>
<td>Block</td>
<td>Block + State</td>
<td>Block + State</td>
<td>Block + State</td>
</tr>
<tr>
<td>Transaction Process</td>
<td>UTXO</td>
<td>UTXO, Smart Contract</td>
<td>Smart Contract</td>
<td>UTXO, Smart Contract</td>
</tr>
<tr>
<td>Consensus</td>
<td>PBFT</td>
<td>PBFT</td>
<td>PBFT</td>
<td>PBFT</td>
</tr>
</tbody>
</table>

**Integrity requirements**

- Correct transaction validity: Yes when \( m \rightarrow c \) if \( c \rightarrow v \) without \( v \rightarrow f \)
- Consensus validity: Yes when \( m \rightarrow v + \) and \( v \rightarrow f \)
- Coercion resistance: Yes when \( m \rightarrow v + \) and \( v \rightarrow f \)
- Finality: Yes (probabilistically) if \( m \rightarrow v + \) and \( v \rightarrow f \)

<table>
<thead>
<tr>
<th>Property</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLT anonymity</td>
<td>Strong</td>
</tr>
<tr>
<td>DLT accountability</td>
<td>Strong</td>
</tr>
<tr>
<td>Consensus robustness</td>
<td>Strong</td>
</tr>
</tbody>
</table>

*1: Not satisfied if special nodes, e.g., maker, notary, or orderer, are in Byzantine failure.
*2: Anonymity by TCert. If subject identity is included in transaction content, same condition as *3 is required.
*3: Under assumption that DLT platform is securely managed and safe from malicious system administrators.

VI. RELATED WORK

Distributed consensus algorithms have been studied for decades in the context of state machine replication [6][8][13][14][15]. Recently, research areas regarding DLT or Blockchain have been gaining attention, especially because of the emergence of permissioned DLT networks. For example, Cachin and Vukolić [8] analyzed consensus protocols and assessed their tolerance to failures. Angelis et al. proposed proof-of-authority (PoA) [18] as an extension of PBFT, specifically aiming at use on Blockchain. Research on the security and privacy aspects of Blockchain has been gaining attention [1], and the privacy issues of virtual currencies are well studied [23]. This paper contributes by identifying general integrity and privacy requirements from real-world use cases and defining their properties in a formal manner.

VII. CONCLUSION

In this paper, we defined security requirements for DLT as a set of integrity and privacy properties and assessed these requirements in four real-life DLT use cases. In addition, we reviewed representative DLT implementations to determine the satisfiability of these properties. As of writing of this paper, most DLT use cases are still in experimental stages, except for virtual currencies. However, as DLT is being adopted for real businesses, stricter and more comprehensive analysis and evaluation of safety will be needed. For many DLT implementations, especially proprietary code, there is a lack of published technical details or source code, so rigid technical assessment is difficult. For DLT to be widely adopted and used, safety properties and evaluation criteria have to be clearly defined to foster open discussion on the safety of DLT.

ACKNOWLEDGMENTS

The authors would like to thank Atsushi Santo, Masafumi Kondo, Go Kondo, and Nobuhide Doi from the Fintech Lab at the Japan Exchange Group (JPX), as well as members of IBM Research – Tokyo for their valuable insights and active discussions.

REFERENCES